Министерство просвещения Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Уральский государственный педагогический университет»
Институт математики, физики, информатики и технологий
Кафедра информатики, информационных технологий
и методики обучения информатике

КОНСТРУКТОР КОМБИНАЦИОННЫХ СХЕМ И КОНЕЧНЫХ АВТОМАТОВ

Выпускная квалификационная работа
бакалавра по направлению подготовки
09.03.02 – Информационные системы и технологии

Исполнитель: студент группы ИСиТ-1601
Института математики, физики,
информатики и технологий
Белоусов А.А.

Руководитель: д.п.н., профессор
кафедры ИИТиМОИ
Стариченко Б.Е.

Допустить к защите
«____» _____ 2020 г.

Зав. кафедрой _____
М.В. Лапенок

Руководитель ОПОП _____
Л.В. Сардак

Екатеринбург – 2020

РЕФЕРАТ

Белоусов А.А. КОНСТРУКТОР КОМБИНАЦИОННЫХ СХЕМ И КОНЕЧНЫХ АВТОМАТОВ, выпускная квалификационная работа: 64 стр., рис. 22, табл. 10, библ. 44 назв.

Ключевые слова: КОМБИНАЦИОННЫЕ СХЕМЫ, КОНЕЧНЫЕ АВТОМАТЫ, ВИРТУАЛЬНОЕ МОДЕЛИРОВАНИЕ, ПРОЕКТИРОВАНИЕ ИС, РЕАЛИЗАЦИЯ ИС.

Предмет разработки – система для виртуального моделирования комбинационных схем и конечных автоматов.

Цель работы – разработка информационной системы, обеспечивающей конструирование и симуляцию работы дискретных комбинационных и последовательных устройств для использования в учебном процессе.

В работе описаны результаты проектирования и программной реализации информационной системы, обеспечивающей конструирование и имитационное моделирование работы дискретных комбинационных и последовательных устройств, для изучения студентами принципов работы устройств для обработки дискретной информации и технической реализации современных электронных устройств.

Система реализована на двух уровнях – клиентском и сетевом. Сетевая часть системы выполнена на программной платформе Node.js с использованием языка JavaScript. Исходные коды клиентской части системы также написаны на языке JavaScript для исполнения в веб-браузере пользователя с использованием библиотек React и G6 для реализации пользовательского интерфейса.

Система внедрена и прошла апробацию в Уральском государственном педагогическом университете; после некоторой адаптации может быть использована в учебной работе других образовательных учреждений.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Изучение принципов работы устройств для обработки дискретной информации необходимо для понимания технической реализации современных электронных устройств. После изучения основ математической логики и теории автоматов, обучающиеся в ходе практических занятий реализуют простейшие цифровые устройства (счётчики, триггеры, сумматоры, и т.д.), используя для этого программы для конструирования и симуляции электронных схем.

В настоящее время существует множество различных программ этого класса, которые успешно развиваются как коммерческие продукты: NI Multisim, Micro-Cap, OrCAD и др. Функциональность подобных программ является избыточной, поскольку рассчитаны на использование высококвалифицированными специалистами. Они имеют коммерческую направленность и высокую рыночную стоимость, несмотря на наличие академических лицензий. Использование данных программных продуктов также требует адаптации в рамках конкретной учебной дисциплины. Существует необходимость в свободно распространяемом программном обеспечении для виртуального моделирования устройств обработки дискретной информации, а именно: комбинационных и последовательных устройств.

Предмет разработки: конструктор комбинационных схем и конечных автоматов.

Цель: спроектировать и реализовать систему для виртуального моделирования комбинационных схем и конечных автоматов.

Для достижения поставленной цели, нужно решить следующие **задачи**:

1. Произвести анализ информационных источников для выявления существующих подходов к моделированию комбинационных схем и конечных автоматов.
2. Проанализировать возможности существующих программных продуктов для виртуального моделирования и обосновать выбор технологий реализации и необходимых программных платформ.

3. В соответствии с техническим заданием провести разработку конструктора комбинационных схем и конечных автоматов.
4. Привести примеры решения задач обработки дискретной информации на основе разработанной системы.
5. Произвести апробацию конструктора используя метод экспертных оценок.

ГЛАВА 1. ТЕХНОЛОГИИ ВИРТУАЛЬНОГО МОДЕЛИРОВАНИЯ

1.1 Виртуальное моделирование логических схем

В начале XX века работавший в России физик-теоретик Пауль Эренфест впервые отметил возможность приложения аппарата алгебры логики, разработанного в середине XIX века ирландским математиком Джорджем Булем, в технике, в частности к электрическим «распределительным сетям» [2, 4]. Эта идея нашла свое воплощение в работах советского физика В. И. Шестакова, а также американского математика Клода Шеннона [3]. Одними из первых объектов применения алгебры логики для решения технических задач были релейно-контактные устройства, состоящие из сотен реле, электронных ламп, электромагнитов и полупроводников. Каждое такое устройство можно было представить в виде схемы состоящей из одних контактов, которые могут находится в двух состояниях – замкнутом и разомкнутом, и такие схемы получили название релейно-контактных схем. Использование алгебры логики в исследовании релейно-контактных схем оказалось возможным потому, что каждой схеме можно поставить в соответствие некоторую формулу булевой логики, и, наоборот, каждую формулу можно выразить с помощью некоторой схемы.

Контактные схемы стали первыми в истории техническими средствами, с помощью которых можно было реализовать булевы функции. В дальнейшем появились и другие устройства, использующие аппарат алгебры логики: оптические, основанные на управляющем воздействии отдельных фотонов света и устройства струйной логики, основанные на явлениях пневматики или гидравлики. Так, например, в 1949 году появился гидравлический компьютер MONIAC для моделирования экономических процессов, а в 90-х годах XX века появились первые прототипы оптических компьютеров, не выдержавшие критики, поскольку при равной функциональности они не обладали той же

компактностью и низким энергопотреблением как компьютеры, основанные на транзисторах [20, 37].

Наиболее актуальным приложением алгебры логики в технике являются *цифровые интегральные микросхемы* (цифровые микросхемы), они широко используются в устройствах электронно-вычислительных машин (ЭВМ), системах автоматики и т. п [15].

Цифровые микросхемы, а также все рассмотренные раннее устройства используются для преобразования и обработки сигналов, изменяющихся по закону дискретной функции. Сигналом называют изменение характеристики материального носителя информации. Например, в контексте электрического тока, сигналом может служить изменение частоты или амплитуды электрических колебаний.

Сигнал, изменяющийся по закону дискретной функции, или же *дискретный (цифровой) сигнал* – это сигнал, являющийся прерывистым и принимающий какое-либо значение из конечного списка возможных значений [12]. В настоящее время распространены *двоичные цифровые сигналы*, они могут принимать два значения – «0» или «1» [29]. Использование двоичных сигналов связано, в основном, с *простотой кодирования*.

При использовании, например, десятичных цифровых сигналов (т.е. используя значения от 0 до 9), потребовалось бы десять уровней напряжения. Предположим, что максимальное значение напряжения определено как 5 Вольт, а минимальное 0,5 Вольт, в таком случае интервал между всеми значениями будет очень небольшой (0,5 Вольт). Если схема подвергается воздействию помех, то напряжение, например, в 3 Вольта может легко обратиться в 4 Вольта, что несомненно вызовет затруднения при кодировании (декодировании) [22]. При использовании двоичных сигналов, разница между минимальным значением и максимальным значением напряжения будет значительной, и помехи не будут оказывать сильного влияния, т.е. двоичные цифровые сигналы обладают также *свойством помехоустойчивости*.

На физическом уровне цифровая микросхема состоит из нескольких транзисторов (или одного транзистора) которые размещают на подложке из полупроводникового материала (её еще называют кристаллом), поэтому про такие схемы утверждают, что они основаны на транзисторно-транзисторной логике.

На логическом же уровне такая схема состоит из набора *логических элементов* – элементов, выполняющих различные логические (булевы) функции (конъюнкцию, дизъюнкцию, инверсию, запоминание и др.) [24, с. 4]. В свою очередь, *логическая функция* – это функция, отображающая одно или несколько высказываний-аргументов в новое высказывание, которое называется значением функции [13]. В свою очередь высказывание — это выражение, про которое однозначно можно сказать истинно оно или ложно. «Истину» и «ложь» обычно интерпретируют как значения булева множества {1, 0}, соответственно [7]. Такие логические элементы называют базисными логическими элементами. Набор элементов выглядит следующим образом:

- НЕ (его также называют инвертором);
- И;
- ИЛИ;
- ИСКЛ. ИЛИ;
- И-НЕ;
- ИЛИ-НЕ;
- ИСКЛ. ИЛИ-НЕ.

Работа каждого такого элемента определяется соответствующей ему логической (булевой) функции, так, например, элементу «И» соответствует функция конъюнкции, элементу «ИЛИ» соответствует функция дизъюнкции, а элементу «НЕ» – функция отрицания.

Среди элементов «ИЛИ», «И», «НЕ» особо выделяется элемент «ИСКЛ. ИЛИ», его работа описывается функцией «исключающее или», её также называют сложением по модулю два, эта функция примечательна тем что её

можно разложить на функции «И», «ИЛИ», «НЕ», так, например, если у нас аргументы функции определены как x1 и x2, то можно представить «исключающее или» в виде ИЛИ(И(НЕ(x1), x2), И(x1, НЕ(x2)). Другими словами, «исключающее или» является композицией функций булевых функций «И», «ИЛИ», «НЕ» или сложной функцией. Из этого следует то, что элемент «ИСКЛ. ИЛИ» состоит из нескольких элементов, но несмотря на это элемент «ИСКЛ. ИЛИ» выделяют в разряд базовых логических элементов [18]. То же самое можно сказать и про элементы «И-НЕ», «ИЛИ-НЕ».

Таким образом, функции «И», «ИЛИ», «НЕ» образуют *базис*, на их основе можно построить другие логические функции. В свою очередь, логические элементы реализующие функции из этого базиса называют *базисными*.

Каждая логическая функция описывается её таблицей истинности [27]. Таблица истинности выглядит следующим образом – в каждом столбце слева находятся значения аргументов, справа находится столбец, содержащий значения функции, при этом значение аргументов в каждой строке таблицы уникально. Пример таблицы истинности для всех базовых логических функций приведен в табл. 1.

Таблица 1.
Таблица истинности для базовых логических функций

x	y	НЕ x	x ИЛИ y	x И y	x ИСКЛ.ИЛИ y
0	0	1	0	0	0
0	1	1	1	0	1
1	0	0	1	0	1
1	1	0	1	1	0

Таким образом, **логический элемент** – это устройство, реализующее функцию, преобразующей набор входных логических сигналов (из множества {0, 1}) в выходной логический сигнал (из множества {0, 1}). Как следует из определения, для элемента необходимо наличие входов и выходов для приема и передачи сигналов.

Логическая схема – логическое представление устройства, обрабатывающего дискретную информацию, совокупность логических элементов, связанных между собой по определенным правилам: выход одного элемента может быть соединен только со входом другого элемента.

Логический элемент также имеет свое графическое представление, это связано с тем, что и логическую схему также можно изобразить графически. Под графическим представлением элемента понимается то, каким образом элемент изображается на схеме.

Существуют несколько общепринятых графических представлений логических элементов:

- Стандарт IEC, использующийся в отечественных ГОСТ и в Европе.
- Стандарт ANSI, использующийся в США.

Сравнение между этими двумя стандартами представлено на рис. 1.

Рис. 1. Стандарты изображений логических элементов

Логическая схема характеризуется количеством входов и количеством выходов.

Входами схемы считают те входы элементов, которые остались свободными, а **выходами схемы**, соответственно, свободные выходы элементов.

Для проверки работоспособности схемы обычно производят набор тестов. Тестирование осуществляется по методу «чёрного ящика»: по очереди берутся значения входов из набора возможных комбинаций, производится подача сигналов и проверка на соответствие ожидаемым значениям на выходах схемы. Очевидно, что для этого нужно каким-то образом подавать входные сигналы на входы схемы. В качестве таких генераторов сигналов обычно используются источники питания (как генераторы постоянного «0» или «1») или тактовые генераторы (генерируют сигналы с заранее определенной частотой). В качестве выходов же используются различные виды индикаторов: светодиоды, цифровые индикаторы, и другие.

Следует отметить, что моменты поступления и выхода сигналов, а также переключения внутренних состояний элементов следуют друг за другом через один и тот же фиксированный промежуток времени, называемый **тактом** [26].

Комбинационными схемами (или комбинационными логическими устройствами) называются такие логические схемы, выходные сигналы которых однозначно определяются сигналами на входах. Как следует из определения они обладают свойством идемпотентности, т.е. при многократной подаче одной и той же комбинации входных сигналов результат всегда будет одним и тем же. При этом в схеме *не должны присутствовать обратные связи*, это обязательное условие.

Конечными автоматами (или схемы с памятью, или последовательные устройства) называются такие логические схемы, выходные сигналы которых определяются не только входными сигналами, но и предысторией их работы. Для хранения этой предыстории используются ***элементы памяти***. Чтобы такие схемы считались последовательными, текущие элементы должны иметь возможность оказывать влияние на работу предыдущих элементов, т.е. в схеме обязательно имеют место ***обратные связи***. Из-за этой особенности такие схемы также называют ***схемами с обратной связью***.

Для изучения процессов, протекающих в дискретных устройствах, исследователями были выработаны соответствующие методы моделирования.

Существуют два ***метода моделирования*** логических схем:

- асинхронный;
- синхронный [19].

Пусть значения сигналов на входах схемы обозначаются как x_1, x_2, …, x_n; значения сигналов на выходах отдельных элементов являются ***состоянием*** элементов и обозначаются как e_1, e_2, …, e_k. Каждый из элементов реализует логическую функцию f. При **асинхронном** моделировании схему описывают системой уравнений вида, где t – номер такта:

$$e_1(t+1) = f_1(x_1(t), x_2(t), \dots, x_n(t), e_1(t), e_2(t), \dots, e_k(t));$$

$$e_2(t+1) = f_2(x_1(t), x_2(t), \dots, x_n(t), e_1(t), e_2(t), \dots, e_k(t));$$

...

$$e_k(t+1) = f_k(x_1(t), x_2(t), \dots, x_n(t), e_1(t), e_2(t), \dots, e_k(t));$$

Перед началом моделирования (при t = 0) необходимо задать состояния всех элементов схемы. Зная состояния элементов схемы и значения сигналов на её входах в некоторый дискретный момент времени t, можно вычислить состояния всех элементов в момент времени (t + 1). Для получения значений выходных сигналов со схемы необходимо производить вычисления состояний до тех пор пока не будет $e_i(t+1) = e_i(t)$ для всех $i = \overline{1,k}$. Другими словами, вычисления производятся до тех пор, пока вся схема не перейдет в устойчивое состояние.

При этом методе порядок записи уравнений, описывающих элементы логической схемы, не имеет значения, при определении состояний элементов на некотором такте в правые части уравнений подставляются состояния, полученные на предыдущем такте.

В качестве примера, произведем асинхронное моделирование схемы сумматора по модулю два (рис. 2). Результаты приведены в табл. 2.

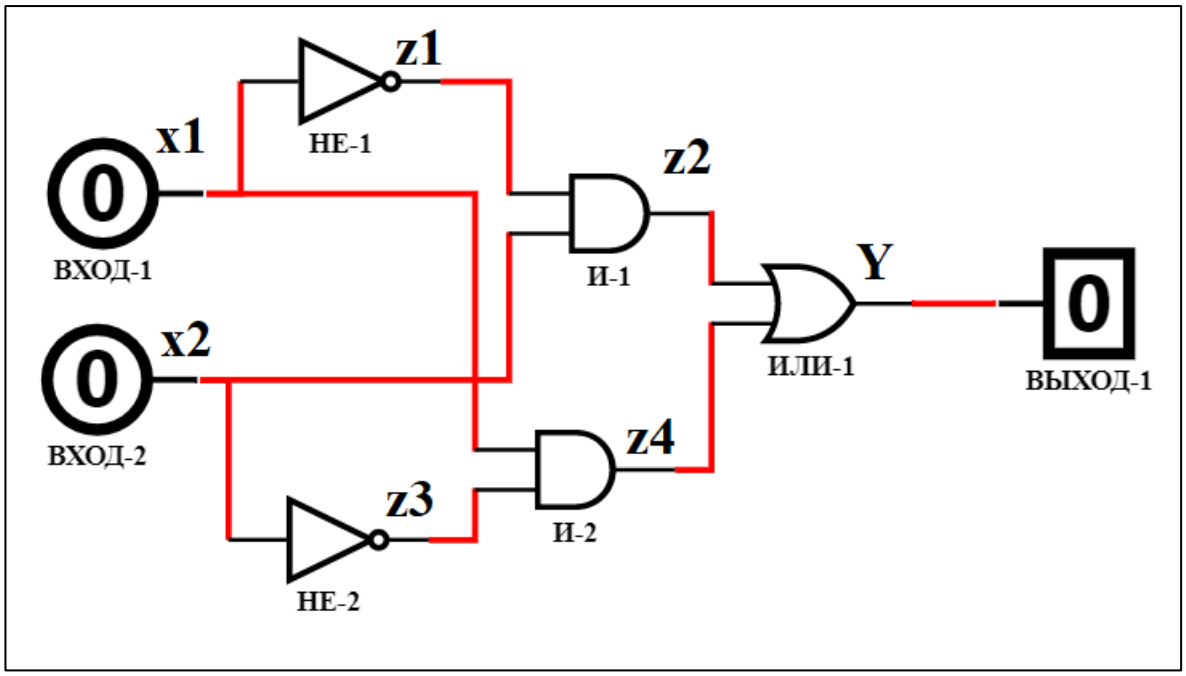


Рис. 2. схема сумматора по модулю два

Таблица 2.

Результаты асинхронного моделирования сумматора по модулю два

Функция	Нач. состояние	Входной набор								
		x_1= 0 x_2 =1				x_1=1 x_2=0				
		Такт								
	0	1	2	3	4	5	6	7	8	9
$z_1 = \text{НЕ}\, x_1$	0	1	1	**1**	**1**	1	0	0	**0**	**0**
$z_2 = z_1 \text{И} x_2$	0	0	1	**1**	**1**	1	0	0	**0**	**0**
$z_3 = \text{НЕ}\, x_2$	0	0	0	**0**	**0**	0	1	1	**1**	**1**
$z_4 = z_3 \text{И} x_1$	0	0	0	**0**	**0**	0	0	1	**1**	**1**
$y = z_2 \text{ИЛИ} z_4$	0	0	0	**1**	**1**	1	1	0	**1**	**1**

Устойчивое состояние схемы наблюдается на четвертом и девятом тактах.

Метод **синхронного моделирования** (или метод Зейделя) предполагает описывать схему следующим образом:

$$e_1(t) = f_1(x_1(t), x_2(t), \dots, x_n(t), e_1(t-1));$$

$$e_2(t) = f_2(x_1(t), x_2(t), \dots, x_n(t), e_1(t), e_2(t-1));$$

...

$$e_k(t) = f_k(x_1(t), x_2(t), \dots, x_n(t), e_1(t), e_2(t), \dots, e_{k-1}(t), e_k(t-1));$$

При вычислении состояний элементов в правые части уравнений подставляются значения входов схемы и состояния элементов, полученные на том же такте, исключая сам вычисляемый элемент, в правую часть подставляется его состояние, полученное на предыдущем такте. Чтобы получить правильное состояние схемы необходимо перед началом моделирования ***упорядочить уравнения***, т.е. записать их в вычисляемой последовательности, поскольку состояние некоторого элемента может быть получено только после вычисления состояний элементов, входящих в уравнение этого элемента.

Для построения вычисляемой последовательности необходимо построить граф вхождения переменных в функции, которые реализуют элементы схемы, а затем провести его топологическую сортировку, т.е. пронумеровать вершины графа начиная со входов. Полученная нумерация и определит вычисляемую последовательность. Данный алгоритм также носит название «ранжирование

элементов». Ранги (номера) элементам присваиваются согласно следующим правилам:

- Входам присваивают нулевой ранг;
- Для элемента, имеющего ранг r, функция обязательно принимает аргумент ранга r-1, любое количество аргументов ранга r-2 и ниже, и никогда не принимает аргументы рангом r и выше.

Для схемы сумматора по модулю два (рис. 2), x_1 и x_2 будут иметь нулевой ранг, z_1 и z_3 – первый ранг, z_2 и z_4 – второй ранг, y – третий ранг.

Вычисления состояний элементов схемы сумматора по модулю два методом синхронного моделирования будут выглядеть следующим образом:

Входной набор $x_1 = 0$ $x_2 = 0$:

$$z_1 = \text{НЕ } x_1 = 1;\ z_3 = \text{НЕ } x_2 = 1; z_2 = z_1 \text{ И } x_2 = (1 \text{ И } 0) = 0; z_4 = z_3 \text{ И } x_1 = \\ = (1 \text{ И } 0) = 0; y = z_2 \text{ ИЛИ } z_4 = (0 \text{ ИЛИ } 0) = 0.$$

Входной набор $x_1 = 0$ $x_2 = 1$:

$$z_1 = \text{НЕ } x_1 = 1;\ z_3 = \text{НЕ } x_2 = 0; z_2 = z_1 \text{ И } x_2 = (1 \text{ И } 1) = 1; z_4 = z_3 \text{ И } x_1 = \\ = (0 \text{ И } 0) = 0; y = z_2 \text{ ИЛИ } z_4 = (1 \text{ ИЛИ } 0) = 1.$$

Входной набор $x_1 = 1$ $x_2 = 0$:

$$z_1 = \text{НЕ } x_1 = 0;\ z_3 = \text{НЕ } x_2 = 1; z_2 = z_1 \text{ И } x_2 = (0 \text{ И } 0) = 0; z_4 = z_3 \text{ И } x_1 = \\ = (1 \text{ И } 1) = 1; y = z_2 \text{ ИЛИ } z_4 = (0 \text{ ИЛИ } 1) = 1.$$

Входной набор $x_1 = 1$ $x_2 = 1$:

$$z_1 = \text{НЕ } x_1 = 0;\ z_3 = \text{НЕ } x_2 = 0; z_2 = z_1 \text{ И } x_2 = (0 \text{ И } 1) = 0; z_4 = z_3 \text{ И } x_1 = \\ = (0 \text{ И } 1) = 0; y = z_2 \text{ ИЛИ } z_4 = (0 \text{ ИЛИ } 0) = 0.$$

Преимущества синхронного метода перед асинхронным заключается в меньшем объеме вычислений; в случае синхронного метода моделирования вычисление состояния схемы происходит всего за 1 такт, а не за несколько, в случае асинхронного метода.

Стоит заметить, что применение синхронного метода в том виде, в котором был описан выше возможно только для комбинационных схем. Если в схеме присутствуют обратные связи построенный граф будет содержать циклы,

следовательно, алгоритм топологической сортировки не будет работать и ранжировать элементы схемы станет невозможно. В этом случае в цепи обратной связи делают разрыв, в этом случае точка разрыва является отдельным входом, которому необходимо задать начальное значение. Разрыв можно сделать явным на схеме, для этого в цепь обратной связи помещают ***элемент задержки***, его также можно интерпретировать как элемент памяти. Это элемент осуществляет задержку входного сигнала на один такт, сигнал поступит на выход лишь на следующем такте.

Данные методы широко используются в программах виртуального имитационного моделирования, на их основе построены существующие алгоритмы симуляции. Однако, следует определить понятие «виртуальное моделирование» и понятие «модель», поскольку дать им однозначное толкование не представляется возможным.

Существует энциклопедическое понятие модели, которое говорит о том, что *модель*, в широком смысле, — это любой образ, аналог (мысленный или условный: изображение, описание, схема, чертеж, график, план, карта и т.п.) какого-либо объекта, процесса или явления («оригинала» данной модели), используемый в качестве его «заместителя», «представителя» [5].

Недостатком этого определения можно считать отсутствие акцента внимания на том, что образ должен отображать существенные для объекта или процесса свойства. В ином случае модель будет перегружена количеством параметров и анализировать её поведение станет намного труднее.

Модели можно разделить на две категории [26, с. 67]:

- Материальные;
- Информационные.

Материальные модели характеризуются тем, что они всегда имеют реальное, физическое воплощение. Примерами таких моделей можно считать глобус (как модель Земли), макет архитектурного сооружения или сделанный из пластика скелет человека.

Информационная модель подразумевает то, что модель представлена в виде информации, т.е. она не является материальной [16]. Эта категория моделей представлена многочисленными результатами когнитивных процессов, строящихся по определенным законам исходя из правил, которые продиктованы изучаемыми и наблюдаемыми объектами, процессами, явлениями. В дальнейшем, они приобретают материальную форму и выражаются в виде чертежа, схемы или рисунка. Все преобразования с такими моделями, в отличие от моделей материальных, осуществляются человеком только в его сознании [14].

Логическая схема, в данном случае, является *информационной моделью*, это абстрактная модель отображающая физическое воплощение реальной цифровой интегральной схемы. Но так как схемы обычно проектируются с помощью прикладных компьютерных программ, логическая схема является также и *компьютерной моделью*, т.е. модель функционирует в некоторой программной среде.

Компьютерная модель и *виртуальная модель* являются одним и тем же понятием, так как «виртуальным» считается нечто, что реализовано программно, симулировано, сымитировано с помощью компьютера [9].

В контексте конструирования цифровых интегральных схем также встречается термин «визуальное моделирование». Например, в книге Роберта Хайнемана «Визуальное моделирование электронных схем в PSPICE», посвященной обучению работе с программой PSPICE путем пошагового построения аналоговых и цифровых схем и их дальнейшего анализа [28]. Стоит отметить, что первая версия PSPICE была выпущена в январе 1984 года [40] и являлась одной из первых программ подобного рода, она задала стандарт того как должны выглядеть программы с графическим интерфейсом пользователя для построения и симуляции работы аналого-цифровых схем.

Хайнеман понимает под моделированием процесс построения схемы и дальнейшей симуляции её работы, от «визуального» здесь только то, что мы

для построения используем графические элементы. Здесь можно согласится с автором, поскольку симулировать функциональность цифровых схем можно, например, используя программную среду Icarus Verilog, которая в качестве описания схемы использует обычный текст, написанный на языке Verilog HDL [34], т.е., не используя никаких графических элементов.

Однако процесс создания модели — это лишь часть процесса моделирования. Корректнее будет рассматривать моделирование как *метод научного познания*, а именно как метод исследования поведения реальных объектов на их моделях [30, с. 34]. Более подробно рассматривает это определение А. А. Братко, который утверждает, что моделирование — это «научный метод исследования различных систем путем построения моделей этих систем, сохраняющих некоторые основные особенности предмета исследования, и изучение функционирования моделей с переносом получаемых данных на предмет исследования» [6, с. 18].

Таким образом, ***виртуальное моделирование*** — это процесс построения модели и её дальнейшего исследования, при этом модель должна способна *симулировать, имитировать* работу некоторого реального объекта или процесса, а также функционировать под управлением конкретной программной среды. В свою очередь, ***модель*** — это новый объект, который воспроизводит наиболее существенные свойства существующего объекта, явления или процесса реального мира.

Процесс моделирования в педагогической деятельности имеет огромное значение. Оно имеет массу преимуществ:

- Использование моделирования при обучении создаёт благоприятные условия для формирования таких когнитивных процессов как классификация, абстрагирование, анализ, синтез и обобщение.
- Моделирование является способом исследования деятельности, а значит, формирования и развития исследовательских компетенций.

- В то время, как студент занимается моделированием, происходит интенсивное овладение информацией о моделируемом объекте (явлении), об отдельных его свойствах. В процессе моделирования исследование объекта-оригинала может проходить значительно быстрее, чем при использовании эмпирических методов.
- Моделирование позволяет получить информацию об объектах и явлениях, непригодных для изучения в аудитории, а также которых нельзя увидеть целиком в окружающем нас мире.
- Моделирование предполагает создание студентом модели самостоятельно, в ходе выполнения практики, а не предъявления её преподавателем в готовом виде.

Основной недостаток метода моделирования в том, что получаемые учебные модели имеют массу упрощений, т.е. студент не получает некоторую часть информации об объекте-оригинале.

Логические схемы обычно рассматривают как систему или модель реально существующих процессов. Необходимость способности моделирования процессов и систем студентами компьютерных специальностей продиктованы федеральными государственными стандартами. Примером может служить ФГОС специальности 09.03.02 «Информационные системы и технологии»:

- способность выбирать и оценивать способ реализации информационных систем и устройств (программно-, аппаратно- или программно-аппаратно) для решения поставленной задачи (ОПК-6);
- способность разрабатывать средства реализации информационных технологий (методические, информационные, математические, алгоритмические, технические и программные) (ПК-12);
- способность проводить моделирование процессов и систем (ПК-5) [23].

Для моделирования логических схем, в учебном процессе используются специальные компьютерные программы для виртуального имитационного моделирования из набора инструментария разработчиков цифровой электроники. Минусы этих программных продуктов следующие:

- *Перегруженность* функционала, обусловленная профессиональной направленностью;
- *Высокий порог вхождения*. Пользователям потребуется некоторый период времени на освоение программ;
- *Проприетарность*. Программные продукты построены на коммерческой основе и не являются свободно распространяемыми;
- *Отсутствие мультиплатформенности* – большинство программ способны работать далеко не на всех популярных операционных системах (ОС).

Таким образом, представляется актуальной разработка программного продукта для виртуального моделирования логических схем, который будет: ***простым*** в освоении для студентов, ***работающим на любых операционных системах*** для персонального компьютера, и что самое главное, являться ***свободно распространяемым***.

Для реализации, в качестве платформы было выбрано веб-приложение, функционирующее в среде веб-браузера. Необходимо произвести анализ существующих программных продуктов, выявить их преимущества и недостатки, и на основе полученных результатов составить техническое задание на разработку.

1.2 Анализ технологий реализаций

Существующие решения для построения и симуляции работы логических схем обладают как преимуществами, так и недостатками.

Мы рассмотрим следующий список программных продуктов:

- Icarus Verilog;

- Multisim Workbench;
- Logic.ly.

Icarus Verilog – это платформа для моделирования и синтеза аналого-цифровых схем. Она включает в себя симулятор (vvp) и компилятор (iverilog) *Verilog HDL* (Verilog Hardware Description Language) – языка описания аппаратуры, использующегося для описания и моделирования электронных систем [1]. Verilog HDL является профессиональным инструментом, он часто используется в проектировании, верификации и реализации аналоговых, цифровых и смешанных электронных систем. В частности, на языке Verilog созданы описания открытых микропроцессоров OpenSPARC (T1, T2), S1 Core и OpenRISC [44].

Принцип работы с Verilog довольно простой. Сначала нужно описать саму схему, а затем тестирующую среду для неё, используя синтаксические конструкции языка Verilog. В терминах Verilog, схема (рис. 3.а) и тестирующая среда (рис. 3.б) являются отдельными *модулями*, они обычно располагаются в отдельных файлах. Затем эти модули нужно скомпилировать, воспользовавшись для этого компилятором, входящим в состав Icarus Verilog. После этого, результат компиляции загружается в основное приложение (рис. 4), где можно наблюдать результат работы схемы в тестовой среде.

```
module half_adder(A, B, H, C);
input A, B;
output H, C;
assign H = A ^ B;
assign C = A & B;
endmodule
```

```
module ha_test;
initial begin
  #10 {A, B} <= 2'b00;
  #10 {A, B} <= 2'b01;
  #20 {A, B} <= 2'b10;
  #20 {A, B} <= 2'b11;
  #50 $stop;
end
reg A, B;
wire H, C;
half_adder ha1(A, B, H, C);
initial
  $monitor("T=%t A=%b B=%b H=%b C=%b",
        $time, A, B, H, C);
endmodule
```

а) б)

Рис. 3. полусумматор и тестовая среда на языке Verilog HDL

Рис. 4. Интерфейс основного приложения Icarus Verilog

Преимущества Verilog заключаются в почти неограниченном спектре возможностей в моделировании аналого-цифровых схем, полнота языка позволяет создать схему любого уровня сложности. Разработчики Verilog HDL при его создании ориентировались на синтаксические конструкции языка C, тем самым сделав его привлекательным для людей знакомых с языками программирования. Само окружение Verilog основано на свободном программном обеспечении, это означает что Icarus Verilog является свободно распространяемым, абсолютно бесплатным и доступным каждому.

Его ***недостатки*** напрямую следуют из его достоинств, перечислим некоторые из них:

- Высокий порог вхождения – чтобы полноценно работать с Verilog, нужно потратить некоторое время на изучение базовых синтаксических конструкций языка и работе с компилятором при сборке модулей.
- Отсутствие наглядности – для ученика, особенно на начальном этапе, крайне важно иметь образ, представление модели. Использование текстового описания схемы лишает нас этой наглядности.
- Отсутствие интерактивности – студент должен сразу же «видеть» изменение поведения модели, при изменении внешней среды, «ощутить»

непосредственный эффект от своих действий. Если нам нужно изменить условия работы тестовой среды необходимо будет изменить код модуля и компилировать все модули заново. Эти «лишние» действия будут также отвлекать студента от непосредственно процесса моделирования.

Следующий программный продукт мы рассмотрим в качестве типичного представителя программ для персональных компьютеров (ноутбуков), попросту говоря «десктопов». **Multisim Workbench**, программа разработанная компанией National Instruments, позволяет моделировать цифро-аналоговые схемы любой сложности. В программе имеется большой набор широко распространенных электронных компонентов. Есть возможность подключения и создания новых компонентов [25], что позволяет производить неограниченную декомпозицию схемы, т.е. объединять целые схемы в единый блок, который рассматривается уже как отдельный элемент схемы более высокого порядка.

Основное ***преимущество*** Multisim Workbench – графический интерфейс пользователя (рис. 5), позволяющий на виртуальном «холсте» размещать элементы схемы, приборы для анализа (вольтметр, амперметр, логический анализатор, осциллограф и другие) и генераторы входных сигналов (цифро-аналоговые источники питания, тактовые генераторы) и соединять их все между собой проводами. Следовательно, с помощью этого инструмента мы можем получить необходимую нам наглядность и интерактивность моделирования по сравнению с Verilog.

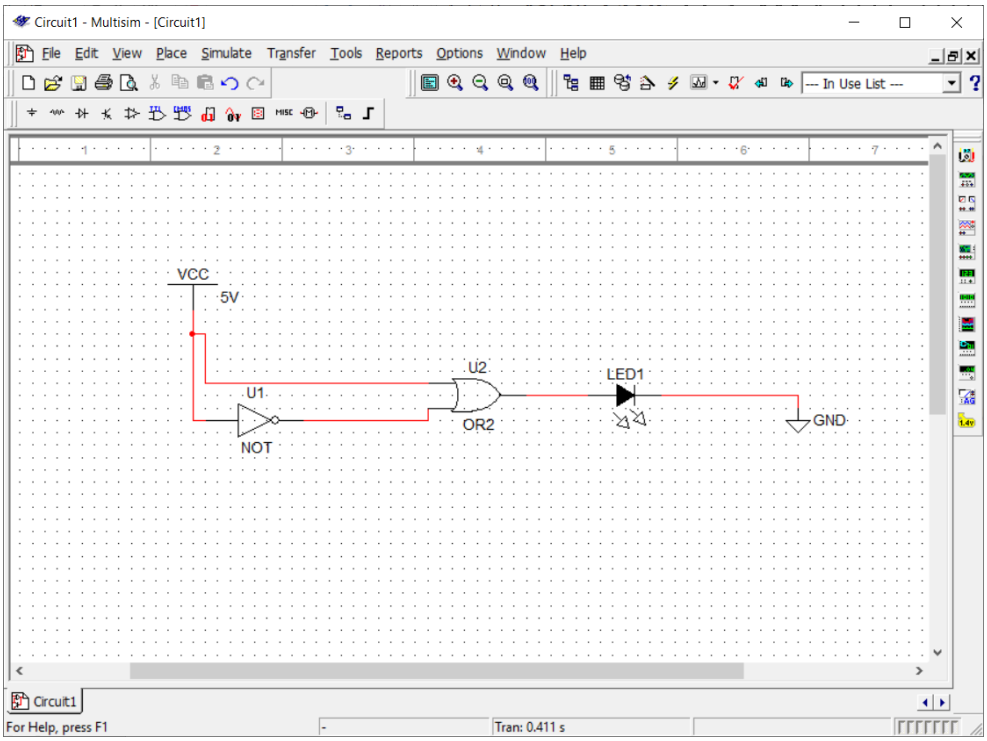


Рис. 5. Интерфейс Multisim Workbench. Пример логической схемы

Перейдем к ***минусам*** данного продукта. Программа является полностью коммерческой, её нельзя получить бесплатно в свободное пользование, доступна лишь 30-дневная демонстрационная версия. Стоимость базовой версии с лицензией на 25 компьютеров для рядового учебного заведения обойдется в 2150 долларов США [8], что является абсолютно неоправданной ценой, профессиональные версии же стоят ещё дороже. Существует также и отдельная студенческая версия продукта Multisim Workbench, её стоимость обозначена в 60 долларов США с лицензией на один компьютер [38], что все равно считается высокой ценой.

Последней мы рассмотрим систему **Logicly**, которая способна моделировать только логические схемы, что, в принципе, следует из её названия. Программа не предназначена для создания сложных цифровых схем, она имеет обучающую направленность, так позиционируют её сами разработчики [36]. Система работает на операционных системах Windows и MacOS, но также имеется и демонстрационная онлайн версия, позволяющая оценить все возможности конструирования и симуляции логических схем.

Онлайн версия Logicly может работать как доступный и бесплатный инструмент для обучения студентов, но в таком случае серьёзным ***ограничением*** окажется *невозможность сохранить схему в файл* (таким образом схема будет работать лишь на компьютере студента) для демонстрации преподавателю. Для его преодоления существуют платные версии продукта, 59 долларов на одного студента, 599 долларов на 20 человек, и версия без ограничений на количество устройств стоимостью 1299 долларов.

Из ***плюсов*** можно отметить:

- Приятный графический интерфейс с широким функционалом (рис. 6).
- Графическое интерпретация логических элементов представлена общепринятыми схемами ANSI и IEC, при знакомстве с более серьезными программными продуктами все элементы будут уже знакомы студенту.
- Онлайн версия может работать в любых операционных системах, т. к. для её работы требуется лишь браузер и Интернет-соединение.

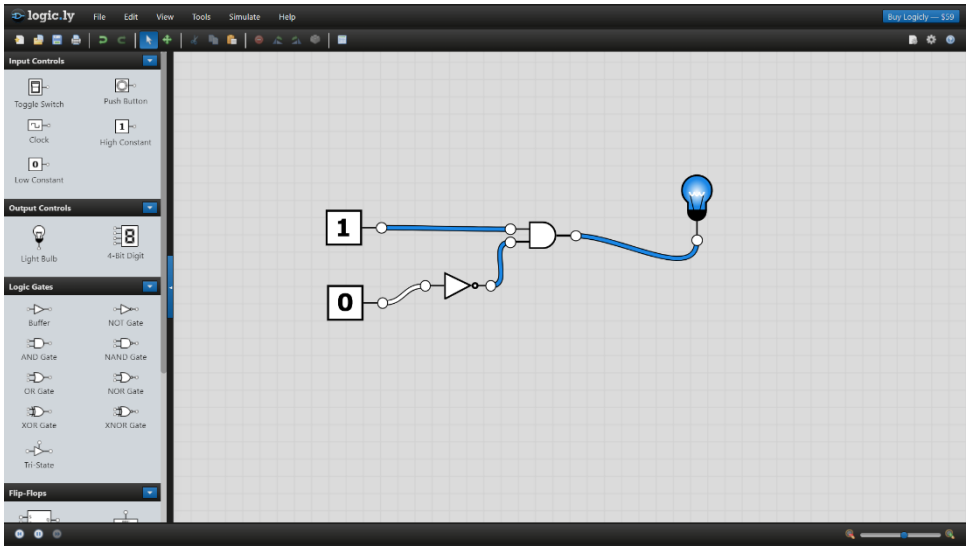


Рис. 6. Пример логической схемы в онлайн версии Logicly

На основе произведенного анализа, выведем основные принципы, которые будут определяющими для реализации собственного веб-приложения:

- Графический интерфейс пользователя;

- Визуальное конструирование схемы с применением графических элементов;
- Низкий порог вхождения;
- Свободная распространяемость.

Веб-приложение всегда предполагает наличие серверной и клиентской части. Необходим дальнейший анализ программных платформ и языков программирования, которые будут использоваться при реализации этих двух частей.

Единственным языком программирования, который способен интерпретировать браузер является JavaScript. **JavaScript** – объектно-ориентированный язык программирования с динамической типизацией, однако прототипирование использующееся в языке создает отличия в работе с объектами по сравнению с традиционными объектно-ориентированными языками. Также, JavaScript имеет целый ряд свойств, которые уже широко используются в функциональных языках — функции как объекты, объекты как списки, каррирование (от англ. curring – преобразование функции с множеством аргументов в набор вложенных функций с одним аргументом), анонимные функции и замыкания — что придаёт языку дополнительную гибкость [17, 35]. Также язык обладает возможностью автоматической сборки мусора, что исключает большинство ошибок при работе с оперативной памятью.

Наиболее актуальным подходом в проектировании клиентской части веб-приложений является ***компонентный*** подход. Компонентный подход предполагает декомпозицию интерфейса на отдельные независимые части – компоненты. Каждый компонент инкапсулируется в отдельном JavaScript модуле. В таком случае, интерфейс представляет собой граф-дерево компонентов. Корневая вершина дерева представляет собой главный компонент, который включает остальные дочерние компоненты, образуя, таким образом, иерархию. Взаимодействие компонентов между собой происходит

следующим образом: компонент который требует ответа на управляющее воздействие (например, клик по кнопке) изменяет свое состояние и/или передает информацию о воздействии родительскому компоненту. Если необходимо сообщить об управляющем воздействии компоненту находящемся в соседнем поддереве, нужно передавать информацию через родительский компонент, из которого можно добраться в соседнее поддерево. Если дерево компонентов огромное, этот факт может приносить неудобства. Для решения этих проблем существует архитектурный подход ***Flux***, который говорит о том, что информацию о произошедшем событии получает каждый компонент, далее компонент самостоятельно решает, как нужно отреагировать на полученное сообщение [32].

Для реализации компонентного подхода существуют отдельные JavaScript библиотеки, наиболее популярные из них: React, Vue и Angular. Возможности этих библиотек схожи между собой, поэтому для разработки веб-приложения был выбран React как наиболее популярный вариант [43]. Особенности React состоят в следующем:

- Виртуальный DOM (от англ. virtual DOM). Все манипуляции с компонентами React выполняет внутри кэш структуры данных в памяти, не используя реальную объектную модель документа, что позволяет быстро вычислять разницу между предыдущим и текущим состояниями интерфейса для оптимального обновления реального DOM.
- JSX (JavaScript XML) – расширение синтаксиса JavaScript, позволяющее использовать похожий на HTML синтаксис для описания структуры компонента, которое включает в себя привычные HTML теги и другие дочерние компоненты.

Поскольку основополагающим принципом приложения является процесс конструирования схемы с помощью графических элементов, существует необходимость в рассмотрении различных вариантов работы с графикой в браузере.

Для работы с простой и несложной графикой в браузере применяют стандартные механизмы DOM, используя в качестве графических элементов HTML элементы, стилизованные с помощью CSS. Минус этого подхода заключается в том, что при большом количестве графических элементов изменение DOM будет происходить медленно, ощутимо замедляя работу всего графического интерфейса веб-приложения. Для решения задач работы со сложной графикой используют Canvas, появившийся в HTML5, а также SVG.

Canvas – элемент HTML5, который предназначен для создания графики с помощью JavaScript [31]. Его используют для рисования графиков и создания анимации. Особенность Canvas состоит в том, что в результате отрисовки получается ***растровое*** изображение и все манипуляции с ним происходят попиксельно. Если необходимо иметь дело с масштабируемой графикой, Canvas окажется неподходящим инструментом.

SVG (Scalable Vector Graphics) – это язык разметки, являющийся расширением языка XML, который используется для описания двухмерной ***векторной*** графики [42]. Его также можно использовать внутри HTML документа.

Для создания изображений в векторной графике используются геометрические примитивы (точки, всевозможные линии, многоугольники, окружности, и т.д.), с помощью которых можно создавать *масштабируемые* изображения, т.е. не теряющие исходное качество при масштабировании.

SVG это технология отрисовки с хранением объектов в памяти (Retained mode graphics). SVG, как и HTML, имеет объектную модель документа (DOM), а также модель событий. Это означает, что при использовании этой технологии для реализации интерактивных действий (таких как перемещение мыши, нажатие клавиш на клавиатуре, и т.п.), разработчик будет затрачивать меньше усилий, поскольку события привязываются непосредственно к элементам DOM. Стоит заметить, что у SVG появляются те же самые проблемы, что и при использовании обычных HTML элементов, а именно: при большом количестве

графических элементов наблюдается замедление работы, поскольку svg-элементы принадлежат тому же DOM, что и обычные HTML элементы.

SVG имеет как обычные атрибуты, так и атрибуты представления. Особенность атрибутов представления заключается в том, что к ним можно применять стили в соответствии со спецификацией применения CSS стилей.

Несмотря на существование отдельных специализированных средств работы с графикой, программисту, при разработке узкоспециализированного программного продукта (например, средства для построения графиков и диаграмм), приходится реализовывать большую часть необходимой функциональности вручную. Для решения этих проблем, программисты-разработчики создают отдельные JavaScript библиотеки, которые применяются именно для решения конкретных задач, перекладывая большую часть работы на себя.

Стоит заметить, что логическую схему можно представить в виде графа, в котором вершинами являются логические элементы, а ребрами «провода», их соединяющие. Для визуализации графов уже существует библиотека, написанная на JavaScript – G6.

G6 – это движок для визуализации графов с открытым исходным кодом, отличающийся простотой и удобством [33]. Он предоставляет набор элегантных решений для визуализации графов и помогает разработчикам создавать приложения для визуализации и редактирования графов. В качестве вершин и ребер графов можно использовать геометрические примитивы (линии, окружности, многоугольники), а также векторные и растровые изображения. При всем этом, библиотека G6 позволяет определять собственные типы вершин и ребер, комбинируя их из уже готовых объектов. Также движок предоставляет хорошо документированный программный интерфейс (API) для произведения манипуляция с объектами графа. Для рендеринга G6 может использовать как Canvas, так и SVG, но разработчики библиотеки рекомендуют использовать первый вариант как наиболее стабильный.

Таким образом, для работы с графическими объектами в конструкторе логических схем целесообразно использовать библиотеку G6.

Рассмотрим следующие платформы для реализации серверной части приложения:

1. *ASP.NET Core.*

ASP.NET Core – фреймворк, набор инструментов и библиотек, разработанный компанией Microsoft для разработки веб-приложений, реализующий шаблон проектирования Model-View-Controller (MVC) (Модель-Представление-Контроллер). ASP.NET Core тесно связан с платформой .NET Core – новой кроссплатформенной реализации платформы уже известной .NET Framework – платформы для разработки ПО для ОС Windows. Большая часть разработчиков уже перешла от старой платформы к новой. Причин тому несколько:

- .NET Core имеет открытые исходные коды, участие в улучшение проекта может принять любой желающий;
- Ощутимое улучшение производительности по сравнению со старой платформой;
- Возможность писать программное обеспечение не только под ОС Windows, но и под системы Linux;

Для разработки ПО на платформе .NET Core используются такие языки программирования как C#, Visual Basic .NET и F#.

2. *Django.*

Django – фреймворк для разработки веб-приложений на языке Python, использующий шаблон проектирования MVC. Большинство инструментов Django для создания приложения являются частью фреймворка, а не поставляются в виде отдельных библиотек, это его отличительная особенность [21].

Фреймворк сразу содержит всю необходимую функциональность для решения большинства задач веб-разработки:

- ORM, позволяющая свободно конвертировать объекты языка Python в сущности баз данных, и наоборот;
- Миграции баз данных;
- Панель администратора;
- Аутентификация и авторизация пользователя;
- Формы;

3. *Node.js.*

Node.js — программная платформа, основанная на движке V8 (который транслирует JavaScript в машинный код), превращающая JavaScript из языка, функционирующего только в веб-браузере, в язык общего назначения [39]. Платформа добавляет возможность JavaScript взаимодействовать с устройствами ввода-вывода, с файловой системой ОС, подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода. Применяют Node.js преимущественно на сервере, таким образом, выполняя роль веб-сервера. В основе Node.js лежит событийно-ориентированное и асинхронное (или реактивное) программирование с неблокирующим вводом/выводом.

Важной частью жизненного цикла ПО является его поддержка. Так как JavaScript на данный момент является самым популярным языком в мире [41], целесообразно разрабатывать клиентскую и серверную часть веб-приложения полностью на этом языке, поскольку один и тот же разработчик сможет полноценно заниматься поддержкой и развитием веб-приложения. Таким образом, для реализации сервера веб-приложения была выбрана платформа Node.js.

Таким образом, представляется оптимальной разработка виртуального конструктора комбинационных схем и конечных автоматов на платформе Node.js с использованием языка программирования JavaScript, при использовании библиотек React.js и G6 для построения пользовательского

интерфейса. Остальные требования к разрабатываемому программному продукту описаны в техническом задании.

1.3 Техническое задание на разработку конструктора комбинационных схем и конечных автоматов

Техническое задание на разработку конструктора комбинационных схем и конечных автоматов основано на следующих документах:

- ГОСТ 34.602-89 [11];
- ГОСТ 19.201-78 [10].

1. Общие сведения.

1.1. Название организации-заказчика

ФГБОУ ВО «Уральский Государственный Педагогический Университет»

1.2. Название продукта разработки.

Рабочее название продукта – «Конструктор логических схем».

1.3. Термины и определения.

Система – веб-приложение «Конструктор логических схем», требования к которому указаны в данном документе.

Схема (логическая схема) – устройство, обрабатывающее дискретную информацию, совокупность логических элементов, связанных между собой по определенным правилам: выход одного элемента может быть соединен только со входом другого элемента.

Элемент (элемент схемы, логический элемент) – элемент, реализующий функцию преобразующую набор входных логических сигналов в выходной логический сигнал, и имеющий собственное графическое представление.

1.4. Назначение системы.

Назначением системы является виртуальное моделирование комбинационных схем и конечных автоматов.

1.5. Плановые сроки начала и окончания работ.

В соответствии с планом выполнения ВКР (01.09.2019 – 19.05.2020).

2. Общие требования к системе.

2.1. Требования к системе в целом.

Продукт должен представлять собой веб-приложение и состоять из двух компонент: серверной и клиентской части взаимодействующих между собой.

2.2. Особенности реализации серверной части.

Сервер должен принимать запросы по протоколу HTTP. Единственная задача сервера – возвращать данные статического веб-сайта (клиентской части веб-приложения) по запросу.

2.3. Особенности реализации клиентской части.

Клиентская часть должна быть реализована в виде HTML документа, работающего в браузере, и состоять из следующих компонент:

- Главной форма;
- Редактор схем.

Главная форма представляет собой диалоговое окно, которое предоставляет пользователю выбор:

1. Загрузить в веб-приложение файл с готовой схемой для дальнейшей работы;
2. Начать работу с редактором «с нуля».

Редактор схем состоит из *панели инструментов*, *палитры элементов схемы*, *области редактирования* на которой располагаются элементы схемы.

3. Требования к функциям (задачам) выполняемых системой.

3.1. Требования к главной форме.

- Пользователь может загрузить файл совершив клик мышью на соответствующий элемент формы или перетащив файл с помощью мыши на соответствующий элемент формы (используя функцию Drag'n’Drop встроенную в браузер).

3.2. Требования к редактору схем.

- Редактор имеет возможность экспорта схемы в файл для последующей загрузки на устройство пользователя;
- Редактор должен иметь два режима работы: режим проектирования и режим тестирования. Пользователь может переключать режимы по своему усмотрению.

3.2.1. Требования к режиму проектирования

- Размещение выбранного на *палитре элементов* элемента на области редактирования;
- Перемещение на области редактирования выбранного элемента (группы элементов);
- Соединение между собой выхода элемента схемы со входом другого элемента «проводом», показывая, таким образом, куда в дальнейшем пойдет сигнал с выхода элемента;
- Удаление элемента/провода из схемы;
- Пользователь может повернуть выбранный элемент схемы на 180°;
- Пользователь может отменить совершенные действия, а также после их отмены применить эти действия снова (система Undo/Redo).

3.2.2. Требования к режиму тестирования

- Возможность установки входов схемы в активное/неактивное состояние;
- Пользователь после установки всех входов может вычислить состояние выходов схемы при заданной конфигурации входов;
- Пользователь должен иметь возможность одновременно привести все входы в неактивное состояние.

4. Дополнительные требования к системе.

4.1. Требования к пользовательскому интерфейсу.

- Система должна корректно отображать пользовательский интерфейс при разрешении экрана от 1024*768 пикселей и больше;
- Интерфейс пользователя должен быть представлен на русском языке;

- *Область редактирования* должна занимать не менее 50% от пространства экрана.

4.2. Поддержка браузеров.

- Система должна корректно работать в следующих браузерах: Edge, Google Chrome, Mozilla Firefox.

4.3. Требования к надежности

- Система должна сохранять состояние редактора схем не менее 15 минут после завершения работы пользователя для последующего восстановления.
- В случае сбоя системы при работе пользователя в редакторе схем, система должна восстановить последнее сохраненное корректное состояние редактора.

5. Перечень сопроводительной документации.

- Руководство пользователя веб-приложения «Конструктор логических схем»

6. Порядок сдачи-приемки продукта.

В соответствии с планом выполнения ВКР.

ГЛАВА 2. РЕАЛИЗАЦИЯ КОНСТРУКТОРА КОМБИНАЦИОННЫХ СХЕМ И КОНЕЧНЫХ АВТОМАТОВ

2.1 Описание алгоритмов и программной реализации

Основная идея реализации клиентской части конструктора схем заключается в следующем:

1. Расширить функциональность движка визуализации G6, поскольку в базовом варианте отсутствуют некоторые необходимые средства для конструирования схем.
2. Спрятать всю работу с библиотекой G6 в отдельном классе, и предоставить внешнему коду лишь упрощенный программный интерфейс.
3. Дополнить созданный класс методами для возможности осуществить симуляцию построенной схемы.
4. Построить пользовательский интерфейс, который будет работать с созданным классом.

Работа с движком визуализации G6 начинается с инициализации графа. Для инициализации необходимо задать ширину и высоту области расположения графа, описание вершин и ребер, а также точку монтирования – HTML элемент внутри которого движок будет заниматься отрисовкой графа.

Движок G6 создает вершины и ребра на основе их объекта конфигурации – объекта, описывающего основные атрибуты. Основные атрибуты объектов конфигурации представлены в табл. 3 и табл. 4.

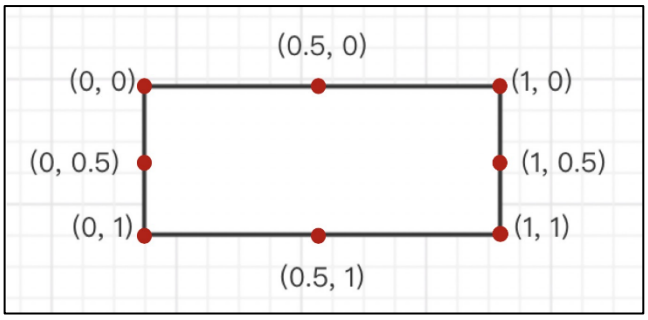
Таблица 3.
Поля объекта конфигурации вершины

Название	***Тип данных***	***Описание***
id	Строка	Идентификатор вершины
x	Число	X координата
y	Число	Y координата
shape	Строка	Название формы, которую будет принимать вершина. По умолчанию – круг

size	Массив/Число	Ширина и высота вершины
anchorPoints	Массив	Массив координат точек, от которых можно прокладывать ребра.
label	Строка	Текстовая надпись (необязательна)

Идентификаторы нужны для быстрого доступа к объекту вершины или ребра. X и Y координаты задают положение вершины на области редактирования. В качестве формы можно указать: прямоугольник, круг, эллипс, ромб, треугольник, звезду, текст, а также готовое изображение (растровое или векторное). Также существует возможность определить собственный тип формы, представляющий собой группу из нескольких готовых форм или формы описанной с помощью контура (path из SVG).

Каждая вершина заключается в ограничивающий прямоугольник (border box), размеры которого задаются атрибутом *size*. Чтобы от вершины можно было прокладывать ребра, необходимо указать координаты точек, от которых ребра будут прокладываться, эти точки разработчики G6 называют «якорными», что отражено в названии атрибута *anchorPoints*. Значения координат точек находятся в диапазоне от 0 до 1, точка (0, 0) находится в левом верхнем углу ограничивающего прямоугольника. Более наглядное представление сущности «якорных» точек и их размещение на вершине



представлено на рис. 7.

Рис. 7. Расположение «якорных» точек вершины

Таблица 4.
Поля объекта конфигурации ребра

Название	***Тип данных***	***Описание***

id	Строка	Идентификатор ребра
source	Строка	Идентификатор начальной вершины
target	Строка	Идентификатор конечной вершины
shape	Строка	Тип линии
sourceAnchor	Число	Индекс в массиве «якорных» точек начальной вершины
targetAnchor	Число	Индекс в массиве «якорных» точек конечной вершины

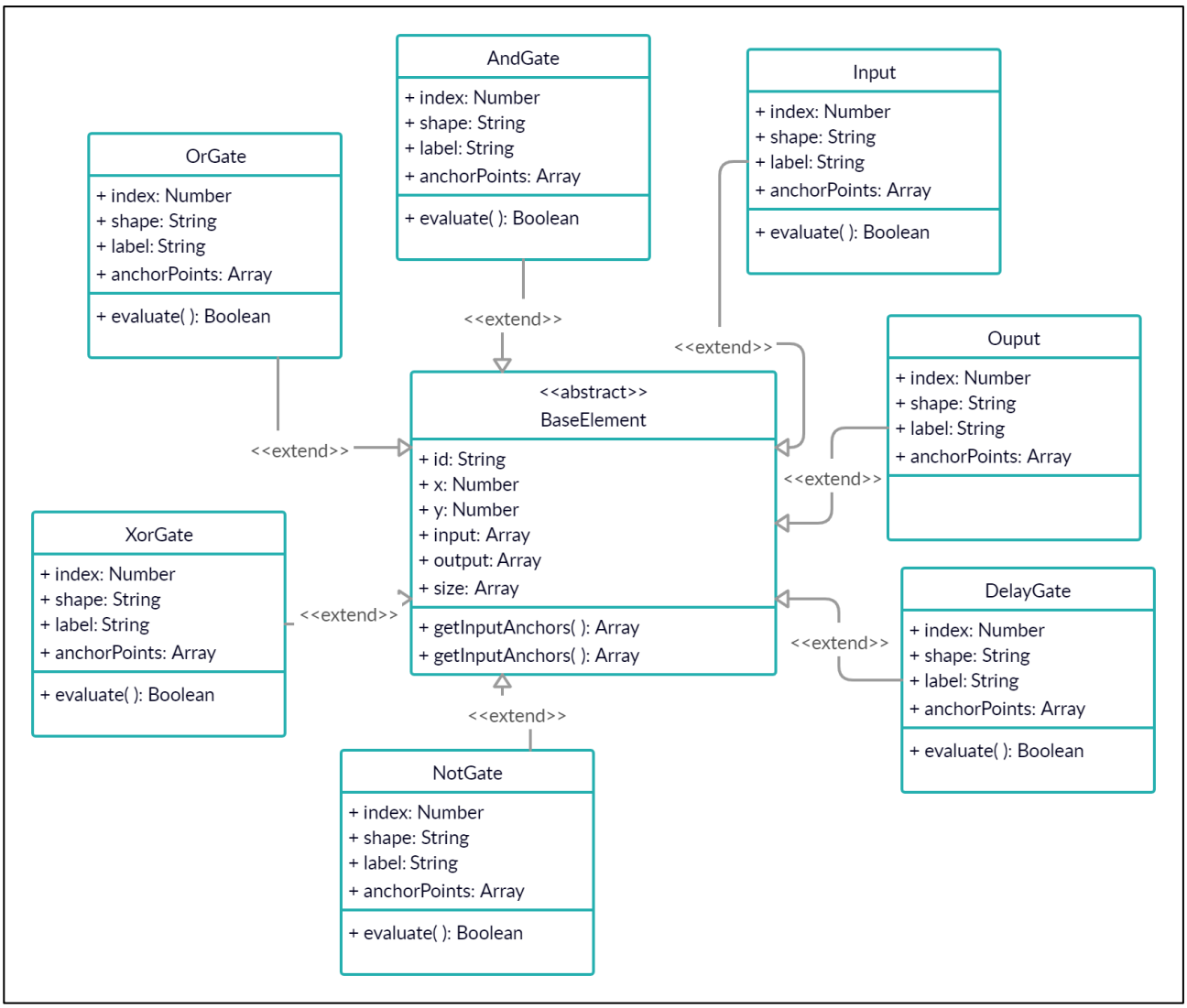
Чтобы построить ребро между вершинами нужно указать идентификаторы вершин *source* и *target*, а также «якорные» точки *sourceAnchor* и *targetAnchor*. Типы линии также определяются готовым набором, и существует возможность определить собственный тип линии. По большому счету, тип линии есть алгоритм того каким образом соединить две точки (например, построив обычную прямую или кривую с заданными параметрами).

Для реализации вершины в виде логического элемента, дополним объекты конфигурации новыми полями и методами, не предусмотренные библиотекой G6, которые мы будем использовать во внешнем коде:

- Поле *input* – массив, предназначенный для установки и хранения входных сигналов элементов.
- Поле *output* – массив, хранящий объекты с атрибутами: ссылка на объект выходного элемента и индекс в массиве *input* выходного элемента (для информации о том, с каким именно входом выходного элемента соединен выход исходного элемента).
- Поле *index* – число, использующееся для генерации идентификатора и подписи *label* элемента. Индексом является порядковый номер элемента определенного типа. Например, при добавлении в пустую схему элементов «И» и «НЕ», индексы у этих элементов будут равны «1», подписи «И-1» и «НЕ-1», идентификаторы «and1» и «not1» соответственно.

- Метод *evaluate* возвращающий результат вычисления логической функции, реализуемой элементом, используя информацию, хранящуюся в массиве *input*.
- Методы *getInputAnchors* и *getOutputAnchors* возвращающие индексы входных и выходных «якорных» точек соответственно.

Объекты конфигурации удобно создавать с помощью классов. Мы определим базовый класс *BaseElement*, от которого будут наследоваться классы конкретных логических элементов. Диаграмма классов логических элементов



представлена на рис. 8.

Рис. 8. Диаграмма классов логических элементов

Также мы определим собственные формы *shape* отдельно для каждого элемента. Форма задается собственным отдельным объектом конфигурации, но нужно его предварительно зарегистрировать перед использованием. Объект обязательно включает атрибут *name* – уникальная строка идентифицирующая тип формы, её используют в поле *shape* объекта вершины и метод *draw* в котором описывается что нужно отрисовать. Регистрация форм происходит перед инициализацией G6 графа посредством вызова метода *registerNode* глобального объекта *G6*.

Для реализации провода соединяющего выходы и входы логических элементов мы зарегистрируем аналогичным образом собственную форму ребра с помощью метода *registerEdge*.

После инициализации графа G6 мы можем получить объект графа и изменять его состояние используя методы данного объекта. Например, метод *addItem* позволяет добавить вершину/ребро, *removeItem* удалить вершину/ребро.

Также после создания графа, он может отслеживать наступление интерфейсных событий, таких как:

- клик мыши по области графа, ребру или вершине;
- движение мыши;
- добавление/удаление вершины (ребра);
- и другие.

При возникновении конфликтов событий, например, клик по области графа совместно с движением мыши необходим, в одном случае для выделения вершин и ребер, а в другом для перемещения области графа существует возможность вынести обработчики событий в отдельные группы и каждую группу вынести в отдельный *режим*. Режимы в G6 позволяют менять поведения редактора изменяя набор групп обработчиков событий.

В нашем случае, в соответствии с техническим заданием предусмотрены два режима – редактирования и симуляции. В режиме симуляции пользователю

запрещено изменять конфигурацию схемы, т.е. добавлять/удалять элементы. В этих двух режимах набор обработчиков будет разный и G6 позволяет переключается между режимами. Чтобы это сделать необходимо:

- Вынести обработчики в отдельные группы по следующему правилу: группа должна реализовывать единственную функциональность (примеры групп: перемещение элемента, добавление провода, перемещение области редактирования).
- Зарегистрировать эти группы с помощью метода *registerBehaviour* глобального объекта G6.
- При инициализации графа в его конфигурации определить поле *modes*, представляющий собой словарь ключами которого является название режима, а значениями массив из групп обработчиков зарегистрированных ранее.

Рассмотрим описание основного класса программы – класса редактора схем *SchemeEditor* (рис. 9), внутри которого происходят все манипуляции с графом, управляемым движком G6.

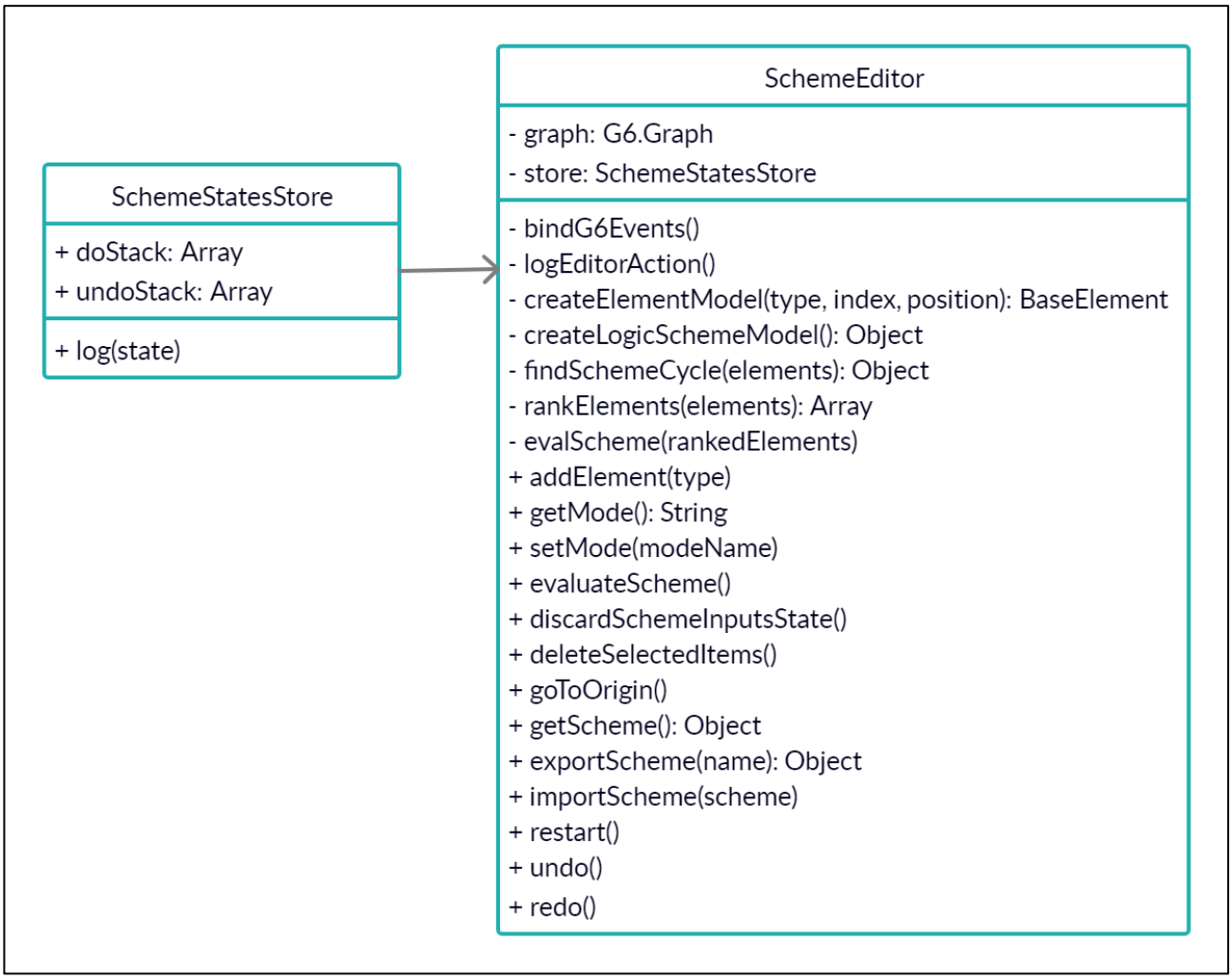


Рис. 9. Описание класса редактора схем

Атрибут *store* содержит в себе хранилище состояний схемы *SchemeStatesStore*, состоянием в данном случае является набор элементов и проводов, а также их расположение на области редактирования. Хранилище необходимо для реализации отмены и повтора изменений схемы (система Undo/Redo). Каждое действие, совершаемое над схемой, приводит к изменению текущего состояния, новое состояние помещается в стек *doStack* в результате вызова метода *log*. На вершине стека *doStack* всегда находится текущее состояние. Отмена какого-либо действия есть отмена текущего состояния и восстановление предыдущего, при этом текущее состояние не удаляется, а снимается со стека *doStack* и перемещается в *undoStack*. Повтор отмененного действия есть снятие со стека *undoStack* состояния и перемещение его на вершину *doStack*. Отметим что новое действие, совершенное после серии

отмен, приводит к очищению стека *undoStack* для предотвращения нарушения цепочки действий.

Алгоритм симуляции схем основан на раннее рассмотренном синхронном методе моделирования (методе Зейделя). Следуя этому методу сначала нужно подготовить граф вхождения элементов, для их дальнейшего ранжирования, – этим занимается метод *createLogicSchemeModel.* Ранги элементам присваиваются в ходе выполнения метода *rankElements.* Однако не всегда представляется возможным произвести ранжирование элементов – в графе могут оказаться циклы, возникающие при присутствии в схеме обратных связей, но только в том случае если в них отсутствуют элементы задержки. Поэтому перед вызовом метода *rankElements* следует проверить наличие циклов вызвав метод *findSchemeCycle*.

Рассмотрим подробнее метод нахождения циклов.

Он представляет собой модификацию довольно известного алгоритма на графах – «обход в глубину». Описать алгоритм можно следующим образом: «Из каждой вершины, которую мы ещё ни разу не посещали, запустим поиск в глубину, который при входе в вершину будет окрашивать её в серый цвет, а при выходе из нее — в чёрный. Если мы пытаемся посетить серую вершину, то это означает, что цикл найден.». В нашем случае граф является ориентированным, так как между элементами (вершинами графа) сигнал идет строго от выходов ко входам элементов, поэтому нет необходимости дополнительно проверять, что текущее рассматриваемое из вершины ребро не является тем ребром, по которому мы пришли в эту вершину.

В приложении используется вариант обхода в глубину без рекурсии, поскольку рекурсивный алгоритм работает лишь на небольших графах, в нашем же случае графы могут быть сколь угодно большими. Нерекурсивный вариант алгоритма позволяет избежать ошибки переполнения стека вызовов, которая происходит, когда рекурсивный обход в глубину запускается на больших графах.

Блок-схема алгоритма поиска циклов в логической схеме, использующегося в нашем веб-приложении представлена на рис. 10. При нахождении цикла алгоритм сразу же завершает свою работу и не занимается поиском других циклов.

После проверки логической схемы на ацикличность, мы можем ранжировать элементы для построения вычисляемой последовательности. Ранги элементам присваиваются в ходе нового обхода графа, при этом использование алгоритма «обход в глубину» не обязательно, можно также построить алгоритм ранжирования и на алгоритме «обход в ширину». В нашей программе используется вариант с обходом в глубину. Блок схема реализованного алгоритма представлена на рис. 11.

После присвоения рангов элементам необходимо сгруппировать элементы по их рангам, результат группировки представлен в виде двумерного массива (массива массивов). По индексу 0 в данном массиве находится массив элементов с рангом 0, по индексу 1 находится массив элементов с рангом 1, и так далее.

Метод *rankElements* сначала ранжирует элементы, а затем формирует и возвращает массив групп.

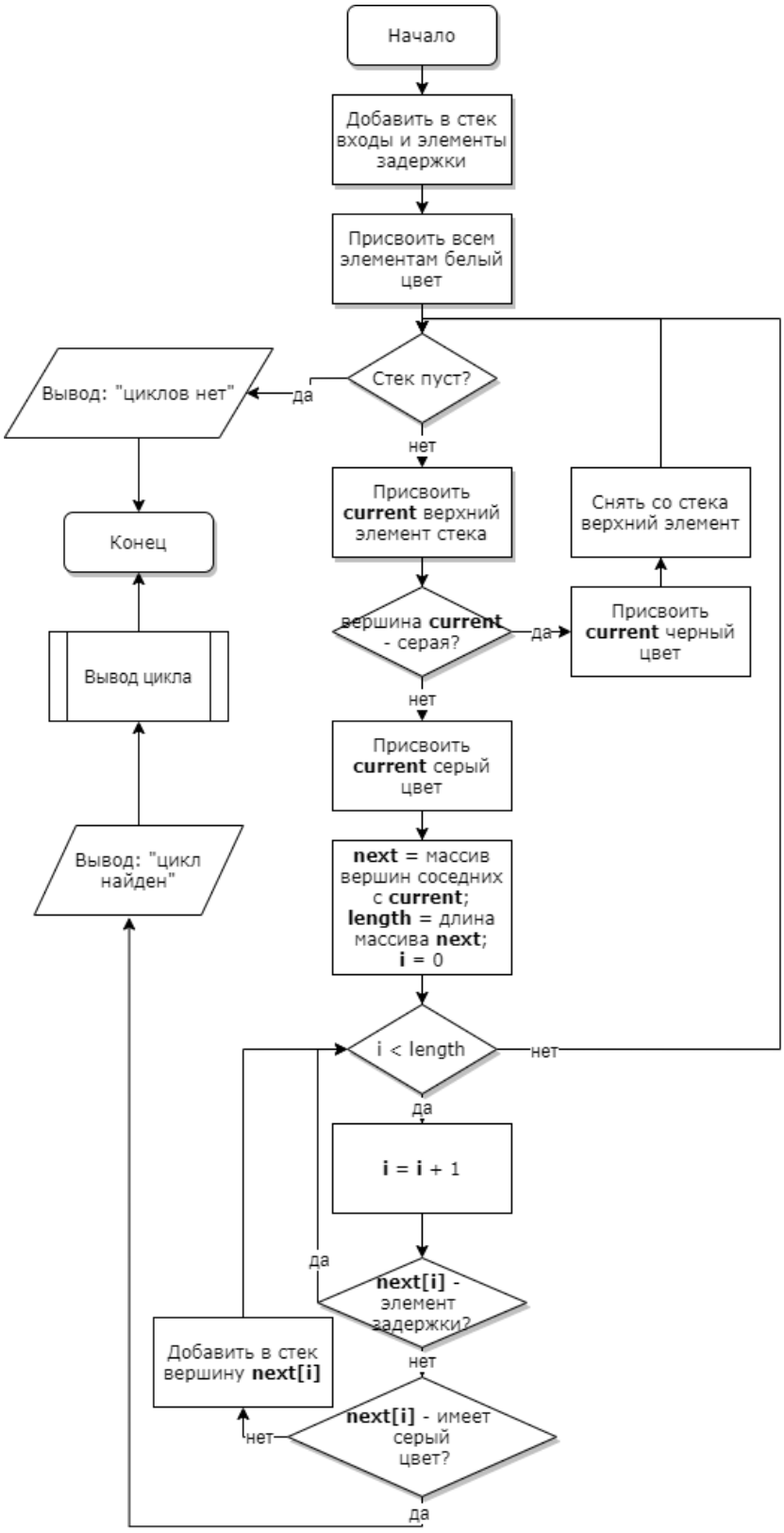


Рис. 10. Блок-схема алгоритма поиска цикла в схеме

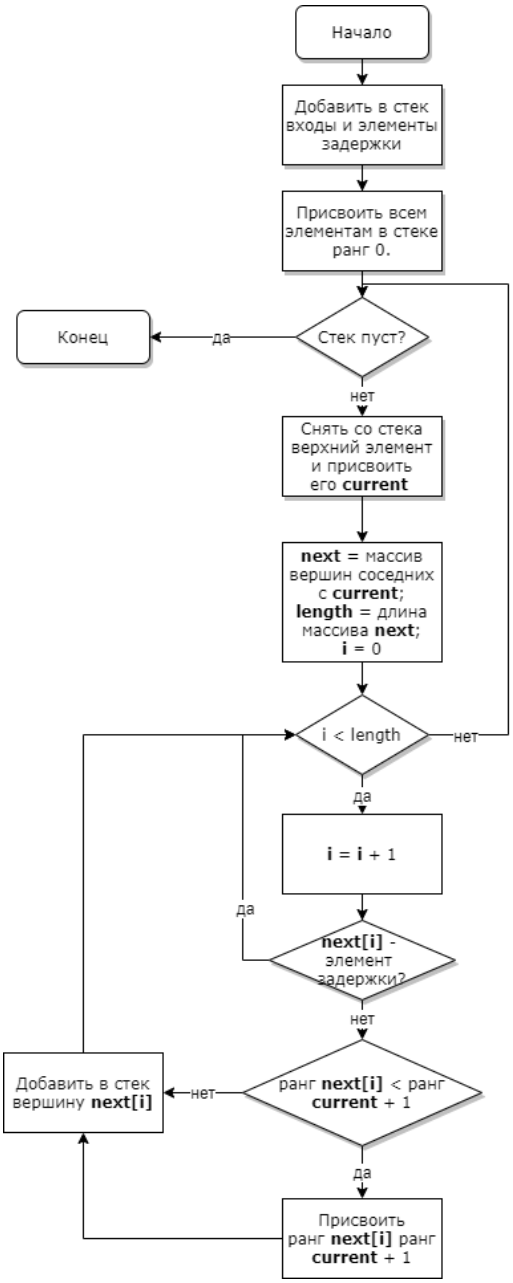


Рис. 11. Блок-схема алгоритма ранжирования элементов

Дополнительно отметим, что массив элементов с рангом 0 имеет определенный порядок: сначала следуют элементы задержки и только потом входы, позже мы дадим объяснение того, почему это важно.

Алгоритм вычисления нового состояния схемы реализован в методе *evaluateScheme,* принимающим на вход двумерный массив ранжированных элементов. В начале для элементов с нулевым рангом производятся следующие вычисления:

1. Вызов у текущего элемента метода *evaluate* для получения значения выходного сигнала элемента.
2. Обход выходных элементов текущего элемента, в ходе которого каждому элементу в массив *input* по известному индексу записывается полученное на предыдущем шаге значение.

Затем аналогичные вычисления производятся для элементов первого ранга, второго и так далее.

Однако может возникнуть ситуация, при которой вычисление входного элемента будет влиять на вычисления элемента задержки, для того чтобы избежать подобных ситуаций на нулевом ранге вычисления сначала производятся именно для элементов задержки.

Пользовательский интерфейс строится с помощью библиотеки React.js. Чтобы использовать React необходимо разбить наш интерфейс на отдельные части и каждую часть представить в виде React компонента. Затем компоненты выстраиваются в иерархию (рис. 12), рендеринг компонентов происходит сверху вниз по дереву. Назначение каждого компонента представлена в табл. 5.

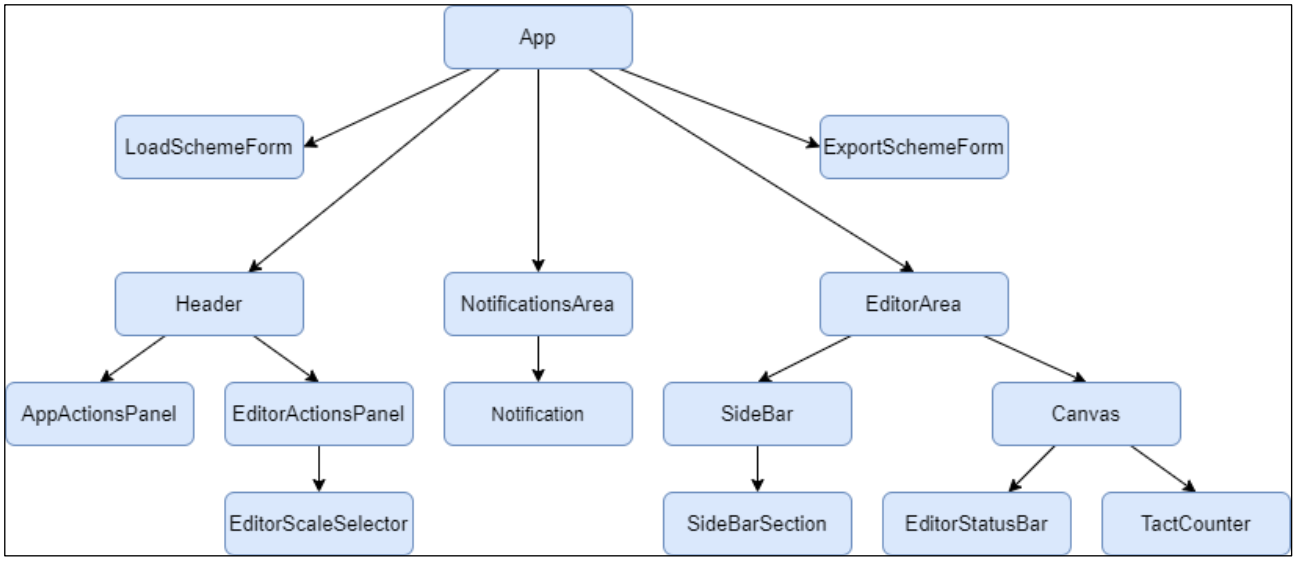


Рис. 12. Иерархия React компонентов веб-приложения

Таблица 5.
Назначение React компонентов веб-приложения

Название	***Назначение***
App	Корневой компонент. Точка сборки всех остальных компонентов.
AppActionsPanel	Панель с действиями относящиеся к общему функционалу приложения (Импорт и Экспорт схемы)
Canvas	Область отображения схемы
EditorActionsPanel	Панель с действиями относящиеся к функционалу редактора схем (удаление элемента, отмена/повтор изменений, переключение между режимами, масштабирование редактора, и другие)
EditorArea	Область редактора схем
EditorScaleSelector	Элемент выбора масштаба редактора из списка
EditorStatusBar	Статус-панель редактора, отображающее координаты расположения курсора и текущий масштаб редактора.
ExportSchemeForm	Форма экспорта схемы
Header	Верхняя панель
LoadSchemeForm	Форма импорта схемы
Notification	Оповещение. Используется для сообщения пользователю о некоторых произошедших событиях в приложении (успешный/неуспешный импорт схемы, возникновение ошибок, и другие)
NotificationsArea	Область размещения оповещений
SideBar	Боковая панель содержащая палитру логических и других элементов схемы
SidebarSection	Секция боковой панели
TactCounter	Счетчик тактов. Отображается в режиме симуляции.

Серверная часть приложения построена на платформе Node.js. Для инициализации сервера используется единственный файл *server.js* в котором импортированы следующие пакеты:

- ***express*** для конфигурации и инициализации экземпляра сервера;
- ***cluster*** для создания нескольких экземпляров серверов для обеспечения максимальной производительности.

Пакеты в Node.js импортируются в самом начале, перед выполнением основной части скрипта, с помощью функции *require*.

Node.js по умолчанию работает в однопоточном режиме, с помощью кластеризации можно добиться того, что запросы к серверу будут обрабатывать все доступные ядра процессора, таким образом осуществляя работу в многопоточном режиме.

```
1  const express = require('express');
2  const path = require('path');
3  const cluster = require('cluster');
4  const numCPUs = require('os').cpus().length;
5  const PORT = process.env.PORT || 8020;
6
7  // Multi-process to utilize all CPU cores.
8  if (cluster.isMaster) {
9    console.error(`Node cluster master ${process.pid} is running`);
10
11   // Fork workers.
12   for (let i = 0; i < numCPUs; i++) {
13     cluster.fork();
14   }
15
16   cluster.on('exit', (worker, code, signal) => {
17     console.error(`Node cluster worker ${worker.process.pid} exited: code ${code}, signal ${signal}`);
18   });
19
20 } else {
21   const app = express();
22
23   app.use(express.static(path.resolve(__dirname, './build')));
24
25   app.get('*', function (request, response) {
26     response.sendFile(path.resolve(__dirname, './build', 'index.html'));
27   });
28
29   app.listen(PORT, function () {
30     console.error(`Node ${isDev ? 'dev server' : 'cluster worker ' + process.pid}: listening on port ${PORT}`);
31   });
32 }
```

Рис. 13. код скрипта server.js

Код скрипта *server.js* представлен на рис. 13. Экземпляр сервера инициализируется с помощью вызова функции *express()* (строка 21). Далее происходит процесс конфигурирования:

- Предоставление доступа к данным клиентской части приложения расположенной в директории */build* с помощью функции *use()* (строка 23).
- Инициализация обработчиков запросов к серверу – все запросы к серверу будут возвращать файл *index.html*, далее веб-браузер на стороне

пользователя самостоятельно загрузит остальную часть клиентского приложения (скрипты, css-стили, изображения).

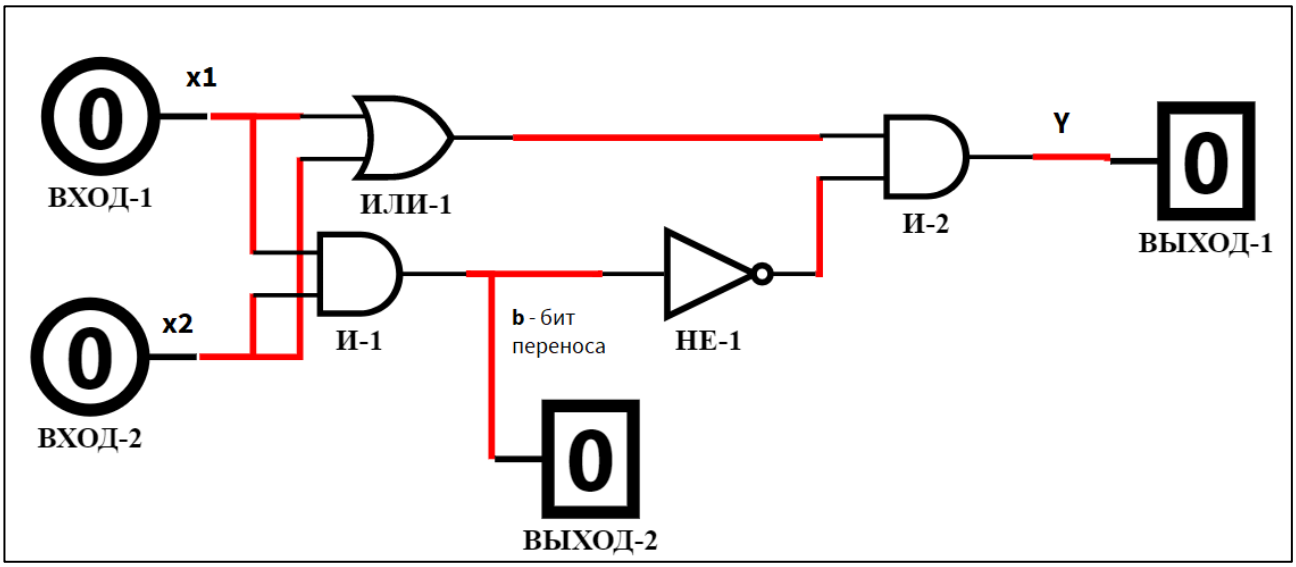
- Указание порта для приема tcp-соединений с помощью вызова функции *listen()* (строка 29).

Таким образом, посредством описанного подхода конструктор комбинационных схем был реализован программно, выполнив все требования, указанные в техническом задании.

2.2 Примеры решения задач с применением конструктора

Мы рассмотрим следующие два класса задач: задачи ***анализа*** и задачи ***синтеза***. Задача анализа предполагает выявление логической функции, которую реализует заданная схема. В свою очередь, задача синтеза состоит в том, что мы теперь выполняем обратную операцию: строим схему по заданной логической функции.

Задача №1: По заданной схеме (рис. 14) определить функцию, построить



таблицу значений.

Рис. 14. Комбинационная схема полусумматора

Решение данной задачи анализа будем производить последовательно, используя для записи логических операций общеупотребимый набор символов (табл. 6).

Таблица 6.

Формы записи логических функции

Название функции	***Символьная запись***	***Буквенная запись***
Конъюнкция	$\wedge$	И
Дизъюнкция	$\vee$	ИЛИ
Отрицание	$\neg$	НЕ
Сложение по модулю 2	$\oplus$	ИСКЛ. ИЛИ

1. На Выход-2, где расположен бит переноса, поступает результат конъюнкции сигналов входов ($b = x_1 \wedge x_2$).
2. На входы элемента И-2 (результирующая функция y) поступает результат инверсии бита переноса и дизъюнкции входных сигналов ($y = (x_1 \vee x_2) \wedge \neg b = (x_1 \vee x_2) \wedge \neg(x_1 \wedge x_2) = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$ – по закону де Моргана).
3. Произведем дальнейшие преобразования: $y = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) =$
$= (x_1 \wedge \neg x_1) \vee (x_2 \wedge \neg x_1) \vee (x_1 \wedge \neg x_2) \vee (x_2 \wedge \neg x_2) =$
$= (x_1 \wedge \neg x_2) \vee (x_2 \wedge \neg x_1) = x_1 \oplus x_2.$

   Таким образом, установлены следующие функции выхода:

$$y = x_1 \oplus x_2;\ b = x_1 \wedge x_2.$$

   Построим таблицу значений (табл. 7).

Таблица 7.

Таблица значений полусумматора

x_1	x_2	y	b
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Перейдем к решению задач синтеза.

Задача №2: По заданной таблице значений (табл. 8) определить функцию обработки и построить схему.

Обратим внимание на то, что значение «1» функция y принимает при x_3 = 1, но только лишь в случае x_1 = 0 и x_2 = 0. Такое же поведение имеет и функция логического «И» – лишь в одном случае из четырех она принимает значение «1». Значит, в третьем наборе значений при x_3 = 1, результат функции над x_1 и x_2 – «1», в остальных случаях где x_3 = 1 результат функции над x_1 и x_2 – «0».

Таблица 8.

Таблица значений логической функции из задачи №2

x_1	x_2	x_3	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Функция над x_1 и x_2, удовлетворяющая данным условиям может быть – $\neg(x_1 \vee x_2)$. Таким образом, выходная функция $y = \neg(x_1 \vee x_2) \wedge x_3$

Перейдем к построению схемы в конструкторе (рис. 15).

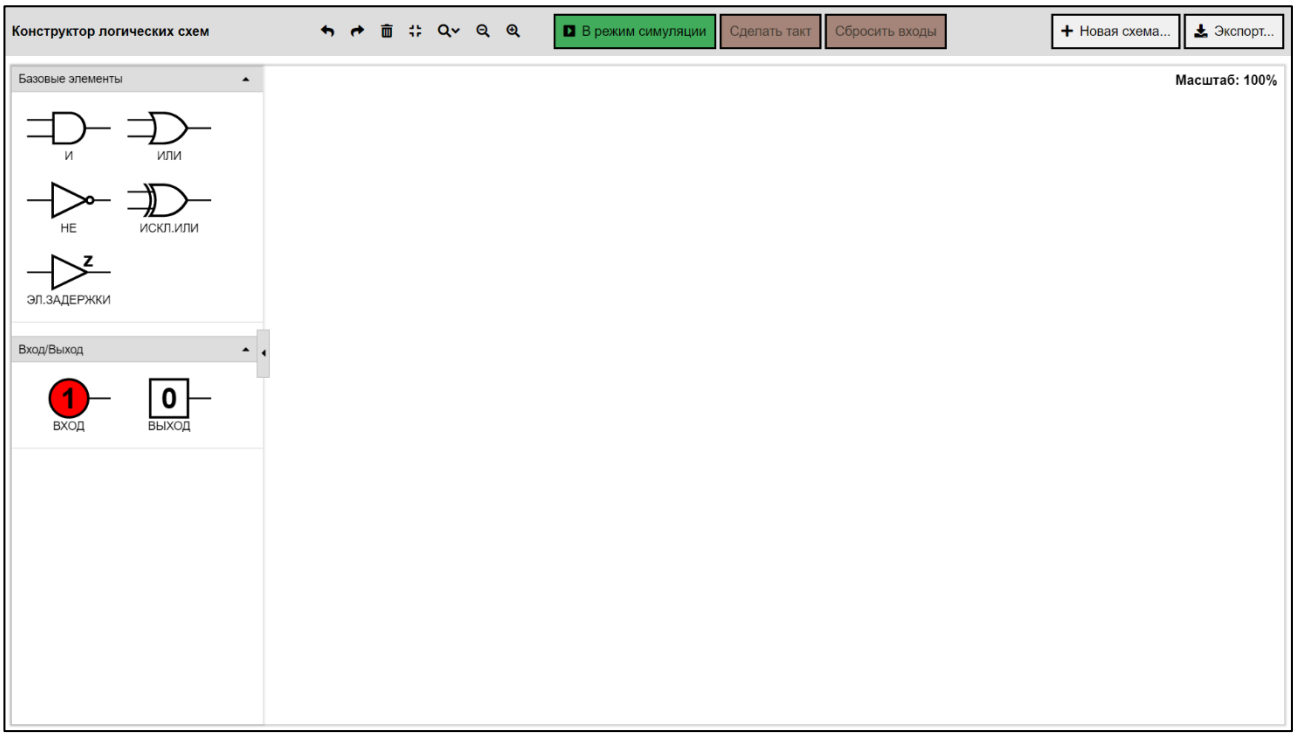
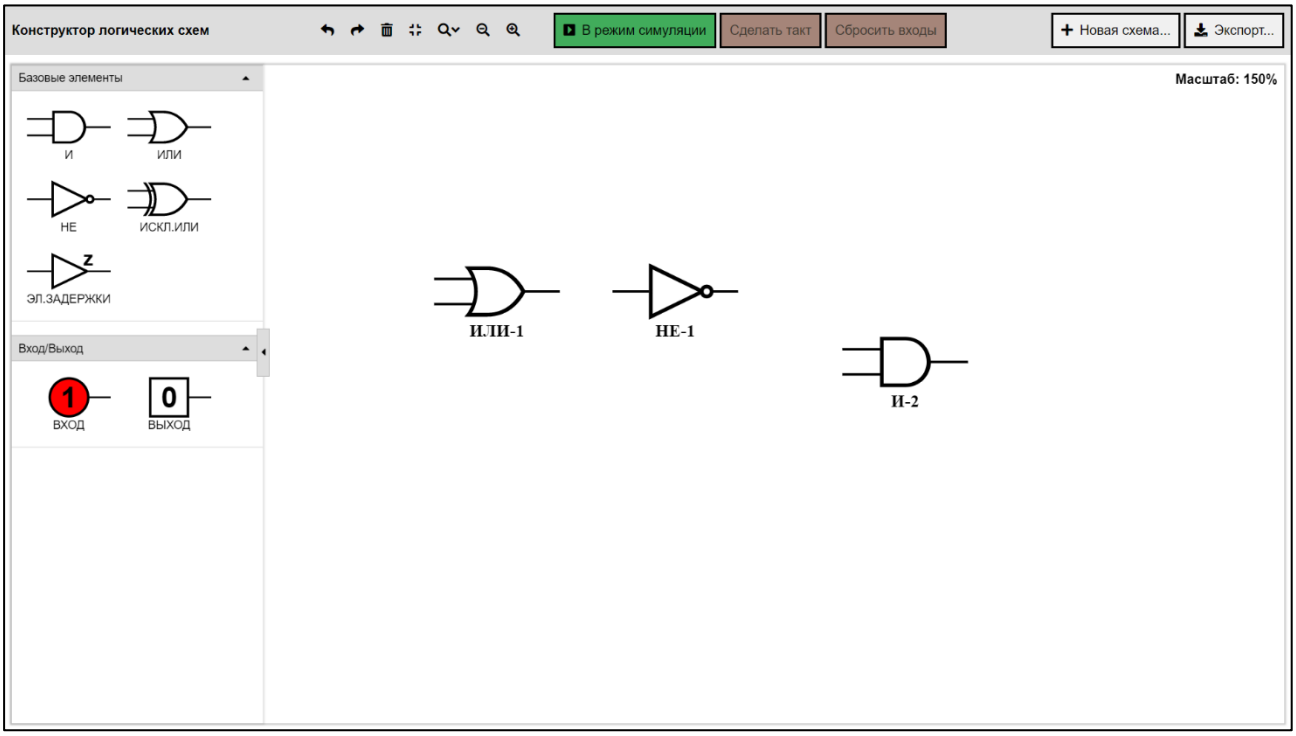


Рис. 15. Конструктор логических схем в режиме редактирования

Разместим на области редактирования необходимые логические элементы. Для добавления элемента переместим курсор мыши на боковую панель слева, и щелкнем левой кнопкой мыши на карточке нужного элемента. Элементы размещаются в одной и той же фиксированной точке, для перемещения элементов разместим курсор мыши над элементом и при зажатой левой кнопке мыши перемещаем курсор вместе с элементом в нужное место.



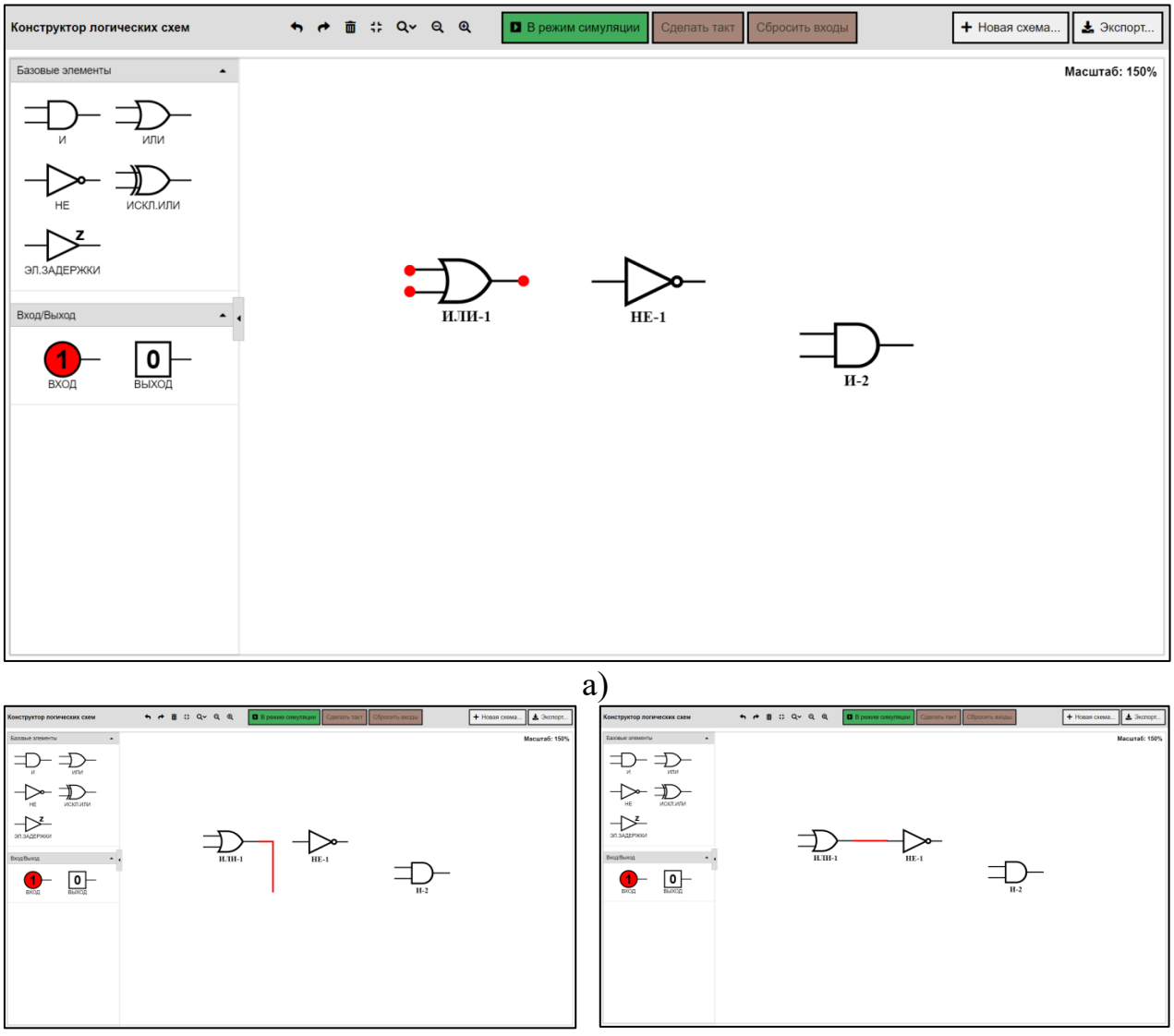
Результат размещения представлен на рис. 16.

Рис. 16. Размещение элементов схемы из задачи №2

Необходимо соединить элементы, для того чтобы можно было передавать логические сигналы. Разместим курсор мыши над коннектором элемента (рис. 17.а), далее по щелчку левой кнопки мыши мы начнем протягивание провода (рис. 17.б), для соединения щелкнем левой кнопкой мыши над нужным коннектором другого элемента (рис. 17.в), провод построит свой путь автоматически. Для удаления элементов или проводов нужно щелкнуть по ним правой кнопкой мыши, также существует и альтернативный вариант: щелкнуть по элементу/проводу левой кнопкой мыши, после этого провод должен

подсветится, а элемент заключится в рамку (рис. 18.а), затем щелкнуть по пиктограмме «мусорный бак» расположенной на верхней панели (рис. 18.б).

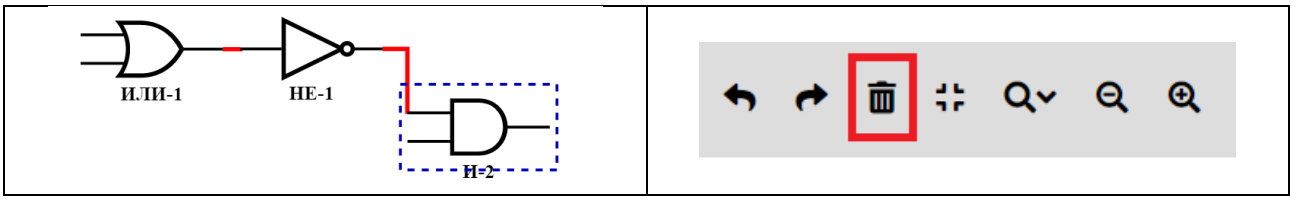
Соединив нужным образом элементы, мы получим искомую схему. Чтобы продемонстрировать её работу необходимо разместить и подключить генераторы сигналов и индикаторы (в программе они обозначены как «ВХОД» и «ВЫХОД»). После этого можно перейти в режим симуляции (нажав на кнопку, расположенной на верхней панели) для тестирования работы схемы (рис. 19).



а)

б) в)

Рис. 17. Соединение элементов в конструкторе



а) б)

Рис. 18. Альтернативный вариант удаления элементов

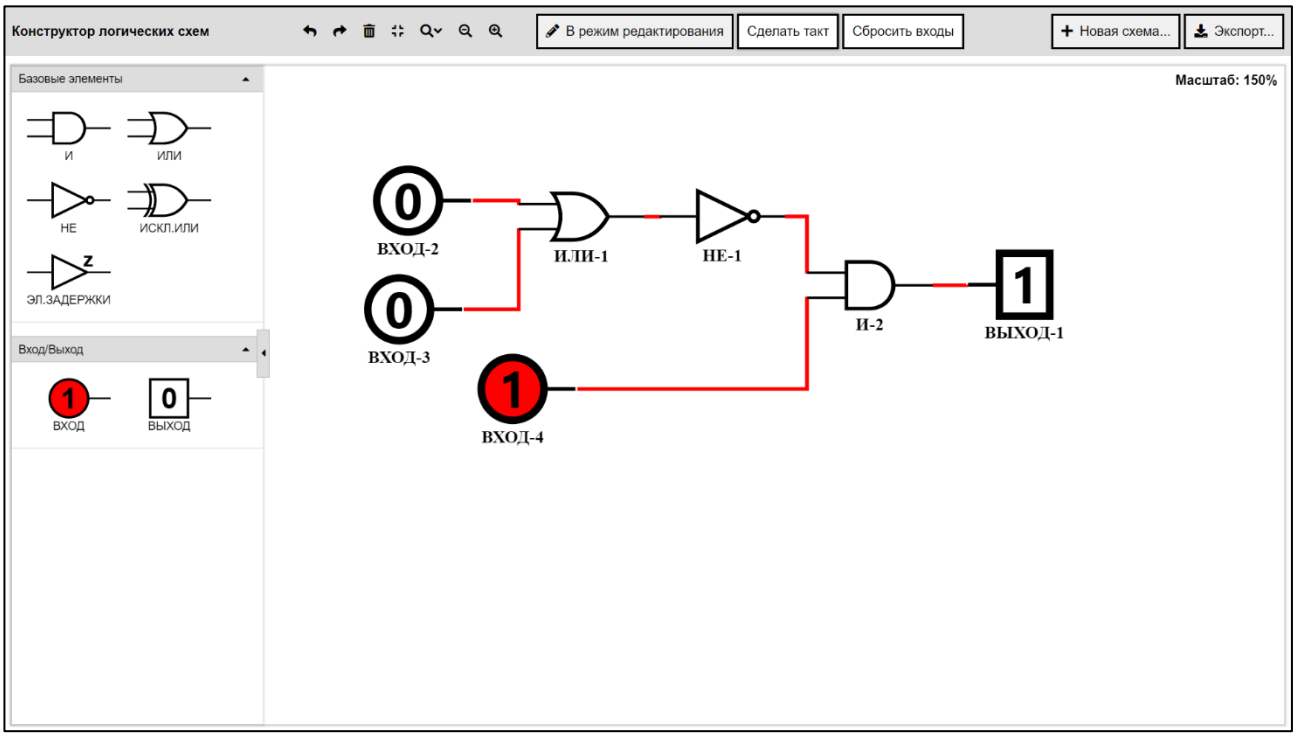


Рис. 19. режим симуляции конструктора схем

После перехода в режим симуляции на верхней панели станут активными кнопки «Сделать такт» и «Сбросить входы». Для переключения состояния входа между «0» и «1» щелкнем на нем правой кнопкой мыши. После установки входов мы можем нажать кнопку «Сделать такт» для вычисления состояния схемы, затем выходы обновят свое состояние, и мы сможем наблюдать результат тактирования (рис. 19). Нажатие на кнопку «Сбросить входы» приведет к установке всех входов в состояние «0».

После проверки корректности собранной схемы, в качестве отчета преподавателю, мы можем экспортировать схему путем нажатия на кнопку «Экспорт…» на верхней панели. В открывшемся диалоговом окне (рис. 20)

задается имя файла схемы, после нажатия кнопки «Ок» начнется загрузка файла. Загрузка файла схемы происходит путем нажатия на кнопку «Новая схема…» верхней панели и щелчка левой кнопки мыши на область «Загрузить схему» в появившемся диалоговом окне (рис. 21), затем средствами операционной системы происходит выбор нужного файла и веб-приложение загружает файл схемы.

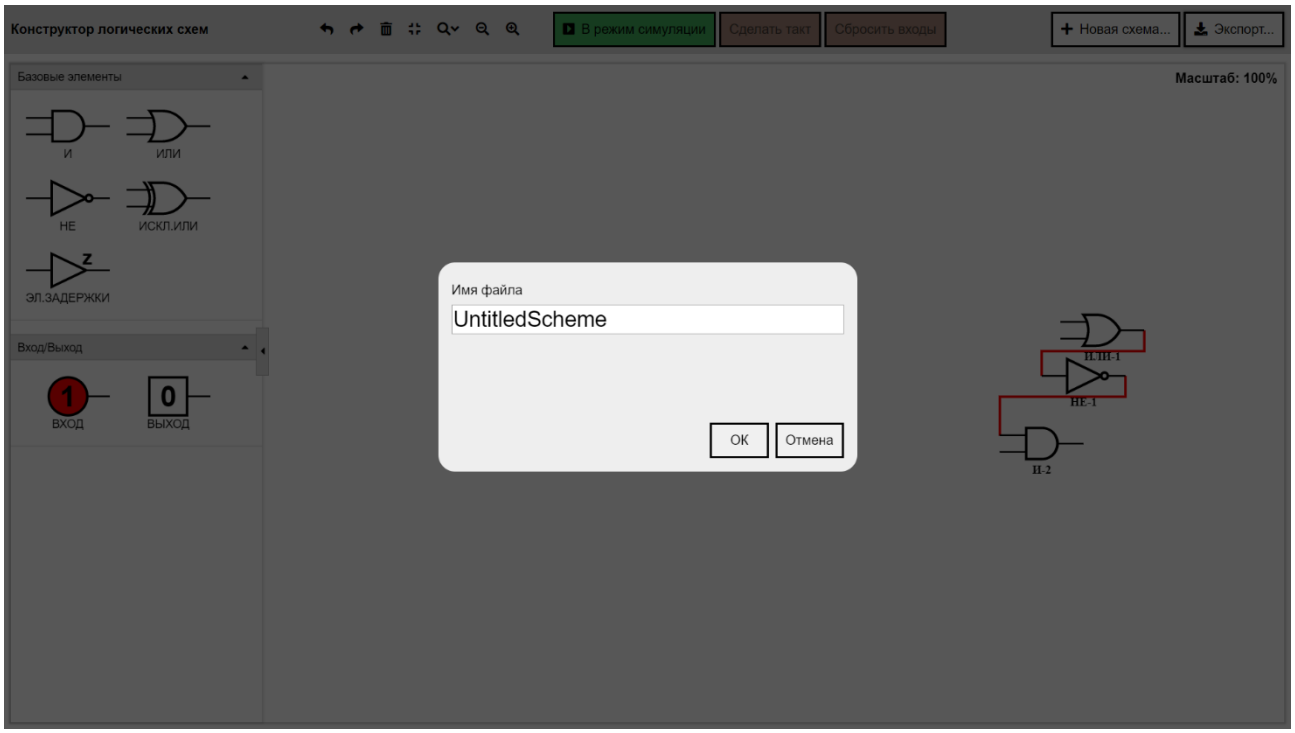


Рис. 20. Диалоговое окно экспорта логической схемы в файл

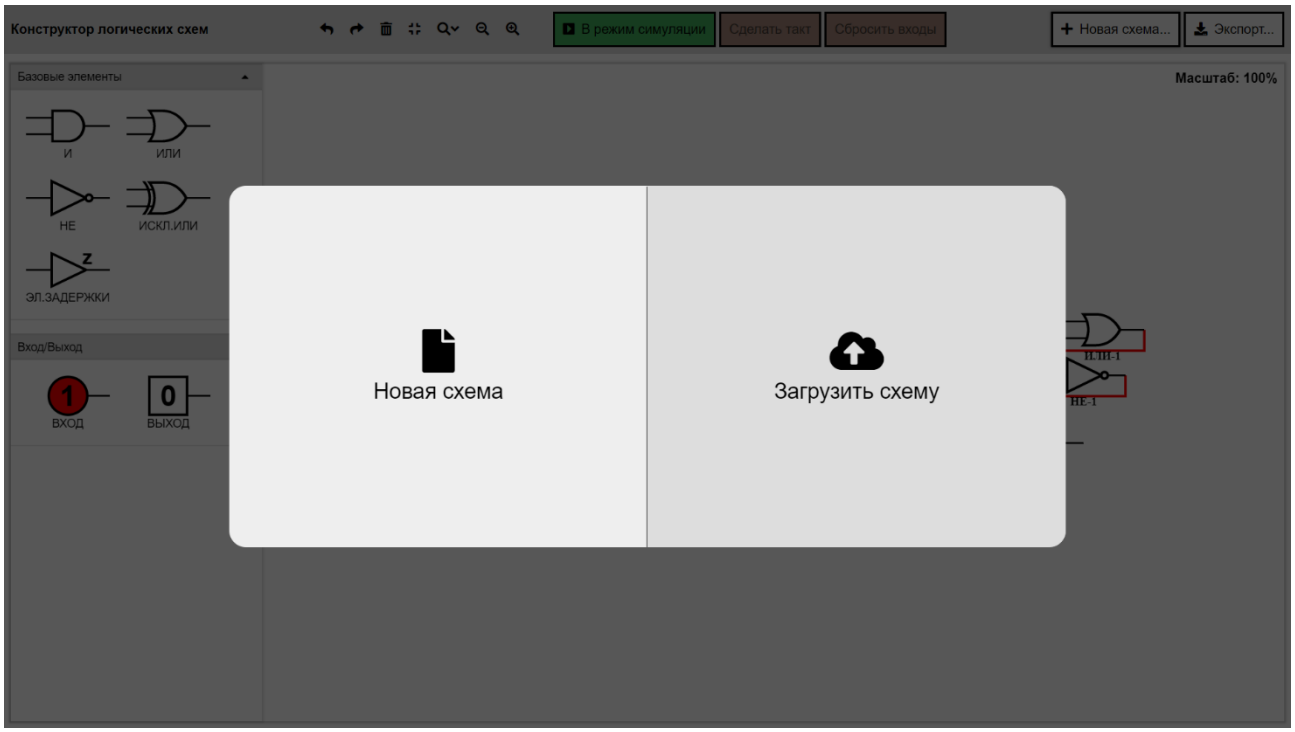


Рис. 21. Диалоговое окно загрузки логической схемы

Задача №3: По заданным функциям $y = x \oplus q`; q = y$ построить таблицу значений и схему конечного автомата. Определить выходное слово если вход: «111001».

Обратим внимание что входами являются только *x*, а выходами только *y*. В данном случае q и $q`$ являются состоянием конечного автомата. Для его представления используются элементы задержки. При этом $q`$ – состояние, полученное на предыдущем такте, а q – на текущем такте. Поскольку выход автомата зависит от его предыдущего состояния, именно $q`$ мы будем рассматривать как отдельный вход.

Построенная таблица значений представлена в табл. 9.

Таблица 9.

Таблица значений конечного автомата из задачи №3

x	***q`***	***q***	***y***
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Построим схему в конструкторе и перейдем в режим симуляции для решения второй части задачи – определения выходной последовательности.

Результат ввода последовательности представлен на рис. 22.а-е.

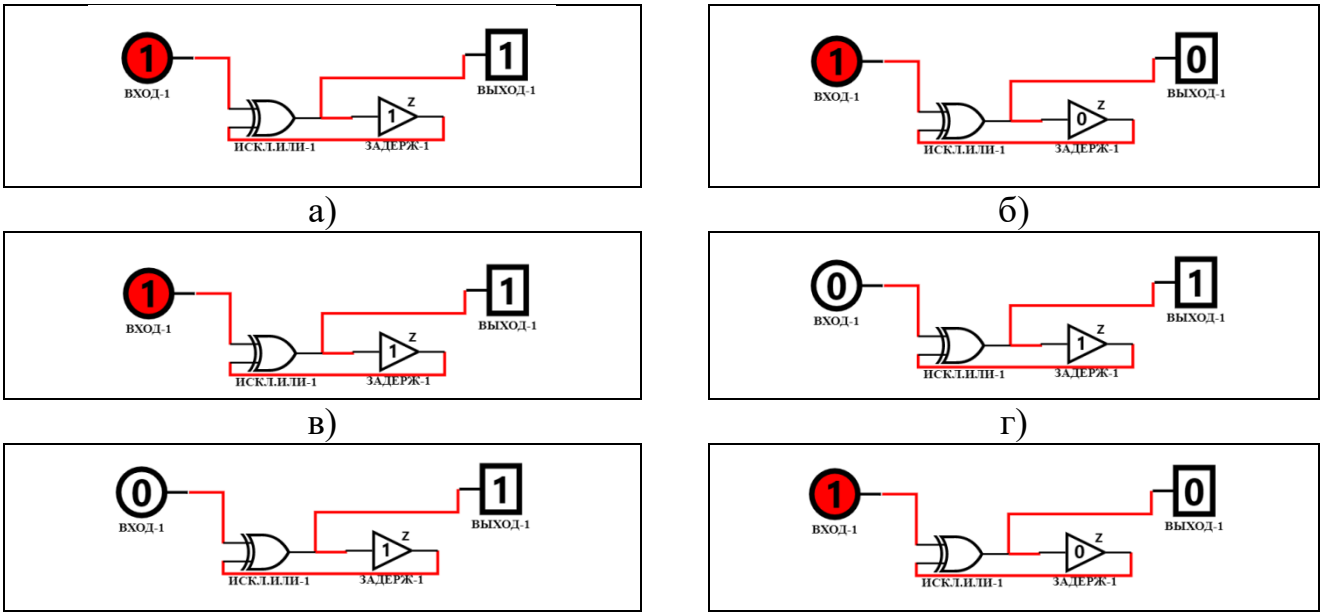


Рис. 22. Моделирование работы конечного автомата из задачи №3

Конструктор работает только с не инициальными автоматами, это означает что начальное состояние автомата всегда имеет значение «0». Обратим внимание что элементы задержки аккумулируют полученный входной сигнал и отображают его. Ответ на задачу: «101110».

Таким образом, были представлены примеры решения различных задач с применением конструктора.

2.3 Результаты апробации

Разработанное веб-приложение «Конструктор логических схем» было развернуто на бесплатном хостинге Heroku. Доступ к приложению свободный.

Работа студентов и преподавателей с приложением основана на обмене файлами:

1. Студент конструирует схему, в соответствии с поставленными задачами.
2. Студент экспортирует созданную схему в файл и пересылает файл преподавателю.
3. Преподаватель импортирует файл студента и проверяет правильность выполнения.

Была проведена апробация конструктора комбинационных схем и конечных автоматов с использованием метода экспертных оценок. В анкетировании приняли участие 6 экспертов-специалистов в области преподавания информатики преподавателей Уральского Государственного Педагогического Университета. Экспертам было необходимо оценить программный продукт ответив на каждый вопрос анкеты, соответствующих отдельному критерию, выставив оценку по пятибалльной шкале (от 1 до 5). Также, экспертам было предоставлено руководство пользователя «Конструктор логических схем». Критерии, представленные в анкете, были следующие:

1. Корректность работы.
2. Удобство пользовательского интерфейса.
3. Скорость загрузки программы в браузере (5-очень быстро, 1-очень медленно).
4. Производительность.
5. Качество руководства пользователя (5-понятно, информативно, 1- непонятно, неинформативно).

Результаты апробации представлены в табл. 10. Согласованность мнений экспертов по каждому из критериев выражается коэфициентом вариации C_v.

Таблица 10.
Экспертные оценки, уровень согласованности экспертов

		Эксперт, оценка								
		1	2	3	4	5	6	Среднее	C_v	Степень согласованности
Критерий	Корректность работы	5	5	5	5	5	5	5,0	0%	Высокая
	Удобство пользовательского интерфейса	4	4	5	5	4	4	4,3	9%	Высокая
	Скорость загрузки программы в браузере	5	5	4	5	4	5	4,7	8%	Высокая
	Производительность	5	5	5	5	5	5	5,0	0%	Высокая
	Качество руководства пользователя	5	5	5	5	5	5	5,0	0%	Высокая

Среди рекомендаций и замечаний экспертов по совершенствованию конструктора были следующие: «Начальное размещение выбранного элемента

в левом верхнем углу поля не слишком удобное», «Серо-коричневый цвет кнопок выглядит мрачновато, нужно доработать», «Желательно адаптировать под мобильные устройства».

Таким образом, по результатам апробации, система после незначительной доработки может быть рекомендована к использованию в учебном процессе.

ЗАКЛЮЧЕНИЕ

На основании проведенного исследования разработана система, обеспечивающая конструирование и симуляцию работы дискретных комбинационных и последовательных устройств для использования в учебном процессе.

В процессе выполнения работы в рамках сформулированных задач было проделано следующее:

1. На основе проведенного анализа информационных источников были выявлены и проанализированы существующие подходы к моделированию комбинационных схем и конечных автоматов, среди которых были выбраны: подход конструирования схем с использованием графических элементов и синхронный метод имитационного моделирования работы схем.
2. В результате оценки функциональных возможностей существующих программных продуктов для виртуального имитационного моделирования был обоснован выбор реализации информационной системы в виде веб-приложения.
3. В соответствии с техническим заданием произведена разработка веб-приложения «Конструктор логических схем» с использованием языка программирования JavaScript и платформы Node.js.
4. Приведены примеры решения задач моделирования логических схем на основе разработанной системы, обеспечивающей самостоятельность работы при решении задач, различную глубину изучения материала, возможность дистанционного взаимодействия с преподавателем.
5. Проведенная апробация показала, что разработанная система хорошо справляется с поставленными задачами. На основании апробации можно заключить, что разработанный конструктор комбинационных схем и конечных автоматов можно рекомендовать к использованию в учебном процессе.

Таким образом, следует считать, что результаты разработки соответствуют всем требованиям технического задания, поставленная цель достигнута. Работа носит законченный характер.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Алексеевский П. И. Обучение студентов архитектуре вычислительных систем с использованием языка Verilog / П. И. Алексеевский. – Текст : непосредственный // Педагогическое образование в России. – 2016. – №7. – С. 131-135.
2. Артемов К. С. Основы цифровой электроники: Учебное пособие / К. С. Артемов, Н. Л. Солдатова. – Ярославль : ЯрГУ, 2013. – 100 с. – Текст : непосредственный.
3. Бирюков Б. В., Борисова О. А., В., Левин В. И. О вкладе В.И. Шестакова в создание логической теории релейных схем / Б. В. Бирюков, О. А. Борисова, В., В. И. Левин. – Текст : непосредственный // Вопросы философии. – 2009. – № 3. – С. 83–92.
4. Бирюков Б. В., Шахов В. И. Первые приложения логики к технике: Эренфест, Герсеванов и Шестаков. От применения логики к расчету сооружений и релейным схемам к логической теории размерностей физических величин / Б. В. Бирюков, В. И. Шахов. – Текст : непосредственный // Логические исследования. – 2007. – № 14. – С. 73–104.
5. Большой Российский энциклопедический словарь. – Репр. изд. – Москва : Большая Российская энцикл., 2009. – 1887 с. – Текст : непосредственный.
6. Братко А. А. Моделирование психики. – Москва : Наука, 1969. – 174 с. – Текст : непосредственный.
7. Булева функция. – Текст : электронный // Википедия: Свободная энциклопедия : [сайт]. – URL: https://ru.wikipedia.org/wiki/Булева_функция. (дата обращения: 21.03.20).
8. Версии продукта Multisim. – Текст : электронный // National Instruments : официальный сайт. – URL: https://www.ni.com/ru-ru/shop/electronic-test-instrumentation/application-software-for-electronic-test-and-instrumentation-category/what-is-multisim/multisim-designers/select-edition.html. (дата обращения: 23.03.20).

9. Виртуальный – Викисловарь. – Текст : электронный // Викисловарь: [сайт]. – URL: https://ru.wiktionary.org/wiki/виртуальный. (дата обращения: 22.03.20).
10. ГОСТ 19.201-78. Единая система программной документации. Техническое задание. Требования к содержанию и оформлению = Unified system for program documentation. Technical specification for development. Requirements for contents and form of presentation : межгосударственный стандарт : издание официальное : дата введения 1980-01-01. – Москва : Стандартинформ, 2010. – 4 с. – Текст : непосредственный.
11. ГОСТ 34.602-89. Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы = Information technology. Set of standards for automated systems. Technical directions for automated system making : утвержден и введен в действие Постановлением Государственного комитета СССР по стандартам от 24.03.89 N 661 : межгосударственный стандарт : издание официальное : дата введения 1990-01-01 / разработан Государственным комитетом СССР по стандартам, Министерством приборостроения, средств автоматизации и систем управления СССР. – Москва : Стандартинформ, 2009. – 12 с. – Текст : непосредственный.
12. Дискретный сигнал. – Текст : электронный // Википедия: Свободная энциклопедия : [сайт]. – URL: https://ru.wikipedia.org/wiki/Дискретный_сигнал. (дата обращения: 21.03.20).
13. Дмуховский С.В. Логические функции. – Текст : электронный : [сайт]. – URL: http://mefestophus.narod.ru/functions.html. (дата обращения: 21.03.20).
14. Землянская Е. Н. Моделирование как метод педагогического исследования / Е.Н. Землянская. – Текст : непосредственный // Преподаватель XXI век. – 2013. – №3-1. – С. 35-43.

15. Интегральная схема. – Текст : электронный // Википедия: Свободная энциклопедия : [сайт]. – URL: https://ru.wikipedia.org/wiki/Интегральная_схема. (дата обращения: 21.03.20).
16. Информационная модель. – Текст : электронный // Википедия: Свободная энциклопедия : [сайт]. – URL: https://ru.wikipedia.org/wiki/Информационная_модель. (дата обращения: 22.03.20).
17. Каррирование функций в JavaScript // Medium : [сайт]. – URL: https://medium.com/webbdev/currying-303cf0be2a0e. (дата обращения: 23.03.20).
18. Логические элементы. – Текст : электронный // Википедия: Свободная энциклопедия : [сайт]. – URL: https://ru.wikipedia.org/wiki/Логические_элементы. (дата обращения: 21.03.20).
19. Моделирование дискретных устройств : методические указания по курсу «Проектирование элементов и устройств систем управления» / В. В. Муханов, А. В. Серегин. – Екатеринбург : ГОУ ВПО УГТУ-УПИ, 2006. – 43 с. – Текст: непосредственный.
20. Оптический компьютер. – Текст : электронный // Википедия: Свободная энциклопедия : [сайт]. – URL: https://ru.wikipedia.org/wiki/Оптический_компьютер. (дата обращения: 21.03.20).
21. Плюсы и минусы Django – Текст : электронный // Хабрахабр : [сайт]. – URL: https://habr.com/ru/post/473042/. (дата обращения 24.03.20).
22. Преимущества двоичного кодирования информации. – Текст : электронный // Справочники от сервиса Автор24 : [сайт]. – URL: https://spravochnick.ru/informatika/preimuschestva_dvoichnogo_kodirovaniya_informacii. (дата обращения: 21.03.20).

23. Приказ Минобрнауки России от 12.03.2015 N 219 (ред. от 09.09.2015) "Об утверждении федерального государственного образовательного стандарта высшего образования по направлению подготовки 09.03.02 Информационные системы и технологии (уровень бакалавриата)". – Текст : электронный // Консультант Плюс : [сайт]. – URL: http://www.consultant.ru/document/cons_doc_LAW_177552. (дата обращения: 22.03.20).
24. Применение интегральных микросхем в электронной вычислительной технике : Справочник / Р. В. Данилов [и др.] ; под редакцией Б. Н. Файзулаева, Б. В. Тарабрина. – Москва : Радио и связь, 1987. – 384 с. – Текст : непосредственный.
25. Разработка и моделирование в программе Electronics Workbench. – Текст : электронный // Схемотехнические решения – Приднестровский портал радиолюбителей : [сайт]. – URL: http://radio-hobby.org/modules/news/article.php?storyid=329. (дата обращения: 23.03.20).
26. Стариченко Б. Е. Теоретические основы информатики: Учебник для вузов. – [3-е изд. испр. и доп.]. – Москва : Горячая линия-Телеком, 2014. – 367 с. – Текст : непосредственный
27. Таблица истинности. – Текст : электронный // Википедия: Свободная энциклопедия : [сайт]. – URL: https://ru.wikipedia.org/wiki/Таблица_истинности. (дата обращения: 21.03.20).
28. Хайнеман Р. Визуальное моделирование электронных схем в PSPICE. – Москва : ДМК Пресс, 2008. – 336 с. – Текст : непосредственный.
29. Цифровой сигнал. – Текст : электронный // Википедия: Свободная энциклопедия : [сайт]. – URL: https://ru.wikipedia.org/wiki/Цифровой_сигнал. (дата обращения: 21.03.20).

30. Ясвин В. А. Образовательная среда: от моделирования к проектированию / В.А. Ясвин. – [2-е изд., испр. и доп.]. – Москва : Смысл, 2001. – 366 с. – Текст : непосредственный.
31. Canvas – Википедия. – Текст : электронный // Википедия: Свободная энциклопедия : [сайт]. – URL: https://ru.wikipedia.org/wiki/Canvas_(HTML). (дата обращения: 24.03.20).
32. Flux – Википедия. – Текст : электронный // Википедия: Свободная энциклопедия : [сайт]. – URL: https://ru.wikipedia.org/wiki/Flux-архитектура. (дата обращения 24.03.20).
33. G6 Graph Visualization Engine. – Текст : электронный // G6 AntV : [сайт]. – URL: https://g6.antv.vision/en. (дата обращения: 24.03.20).
34. Icarus Verilog – Википедия. – Текст : электронный // Википедия: Свободная энциклопедия : [сайт]. – URL: https://ru.wikipedia.org/wiki/Icarus_Verilog. (дата обращения: 22.03.20).
35. JavaScript – Википедия. – Текст : электронный // Википедия: Свободная энциклопедия : [сайт]. – URL: https://ru.wikipedia.org/wiki/JavaScript. (дата обращения 23.03.20).
36. Logicly – Main Page. – Текст : электронный // Logic.ly : [сайт]. – URL: https://logic.ly/. (дата обращения: 23.03.20).
37. MONIAC – Википедия. – Текст : электронный // Википедия: Свободная энциклопедия : [сайт]. – URL: https://ru.wikipedia.org/wiki/MONIAC (дата обращения: 22.03.20).
38. NI Multisim Student Edition, National Instruments. – Текст : электронный // Studica : [сайт]. – URL: https://www.studica.com/multisim-student-edition.html. (дата обращения: 23.03.20).
39. Node.js – Википедия. – Текст : электронный // Википедия: Свободная энциклопедия : [сайт]. – URL: https://ru.wikipedia.org/wiki/Node.js. (дата обращения: 24.03.20).

40. PSpice – Википедия. – Текст : электронный // Википедия: Свободная энциклопедия : [сайт]. – URL: https://ru.wikipedia.org/wiki/PSpice. (дата обращения: 22.03.20).
41. Stack Overflow Developer Survey 2019. – Текст : электронный // Insights Stack Overflow : [сайт]. – URL: https://insights.stackoverflow.com/survey/2019/#technology-_-programming-scripting-and-markup-languages. (дата обращения: 24.03.20).
42. SVG – Википедия. – Текст : электронный // Википедия: Свободная энциклопедия : [сайт]. – URL: https://ru.wikipedia.org/wiki/SVG. (дата обращения: 24.03.20).
43. The State of JavaScript 2018: Front-end Frameworks – Overview // State of js : [сайт]. – URL: https://2018.stateofjs.com/front-end-frameworks/overview/. (дата обращения 24.03.20).
44. Verilog – Википедия. – Текст : электронный // Википедия: Свободная энциклопедия : [сайт]. – URL: https://ru.wikipedia.org/wiki/Verilog. (дата обращения: 23.03.20).